MULTISTAGE STOCHASTIC PROGRAMMING

A Dissertation
Presented to
The Academic Faculty

By

Lingquan Ding

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

August 2020

MULTISTAGE STOCHASTIC PROGRAMMING

Approved by:

Dr. Alexander Shapiro, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Nils Löhndorf
Luxembourg Centre for Logistics and Supply Chain Management
University of Luxembourg

Dr. George Lan
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Andy Sun
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Enlu Zhou
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: April 6, 2020

To my parents, for their love and encouragement

and to the memory of Dr. Shabbir Ahmed.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to many people. First and foremost, my deepest gratitude goes to my advisor, Dr. Alexander Shapiro. He has played an instrumental role in my life at Georgia Tech. Four years ago, I took the time series class taught by Dr. Shapiro. Intrigued by his academic contribution and personality, I started a journey with him in operations research. Without his incredible guidance, unreserved support and consistent encouragement, this journey would not be so fruitful and this dissertation would not exist.

My gratitude also goes to my co-advisor, Dr. Shabbir Ahmed in memorial. His intellectual insights and dedication to research have inspired me a lot. I already miss him a lot during my last year of my Ph.D. But I believe his legacy lives forever and his contribution will be always remembered.

I would like to thank Dr. Nils Löhndorf, Dr. George Lan, Dr. Andy Sun and Dr. Enlu Zhou for serving on my thesis committee and for their helpful comments and insightful feedback. I would also like to thank amazing ONS colleagues including Filipe Goulart Cabral and Joari Paulo da Costa for their support of numerical studies and implementations.

I feel extremely privileged to have the opportunity to study at ISyE. I would like to thank Dr. Hayriye Ayhan, Dr. Santanu Dey, Dr. Anton Kleywegt, Dr. Arkadi Nemirovski, Dr. Yajun Mei and Dr. Shijie Deng for their excellent courses. I would also like to thank my officemates, Mohamed El Tonbari, Xiaoyi Gu, Emma Johnson, Zhe Zhang, Alexandre Velloso, my roommate Yan Li and many other friends at ISyE.

Finally, I would like to thank my parents. They are always my backbone. Without them, I could not be at this point.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

There are many practical problems where one has to make decisions sequentially based on data (observations) available at the time of the decision. Trying to make such decisions under uncertainty in some optimal way, looking forward in time, leads to the area of multistage stochastic optimization. In this thesis, we develop methodologies, algorithms, and a software package for large-scale multistage stochastic programming problems with applications in energy, airline and finance.

In the first part of the thesis (chapter 2), we suggest a standardized procedure to solve multistage stochastic programs. The procedure has four steps. The first step is to model the underlying data process and build the true problem. In many applications, the underlying data process has a Markovian structure. We discuss two ways to embed the data process into the optimization problem. The second step is to discretize the true problem. Various approaches of discretization are proposed. The third step is to solve the discretized problem. A powerful tool to do that is the so-called stochastic dual dynamic programming algorithm (SDDP). Computational aspects related to this algorithm are discussed. In particular we discuss three types of stopping criteria of the algorithm. The fourth step is to evaluate the obtained policy on the true problem. This final step is of crucial importance, which is often ignored in the literature. Extensions of the methodology to risk averse and mix-integer settings are also considered. We also develop another class of algorithms based on equivalent linear programming formulations that do not rely on stochastic programming.

In chapter 3, we demonstrate a new software package, MSPPy, for multistage stochastic programs based on the aforementioned methodologies. Since the popularity of the SDDP algorithm, many open-source software packages have sprung up and are able to solve a wide variety of problems. But all of them were built for the case when the underlying data process has finite number of realizations (scenarios). The newly built software package is the first one to consider problems with continuous distributions of the data processes.

In the second part of the thesis (chapter 4), we propose a new variant of the SDDP algorithm, referred to as periodical SDDP. Some real-world problems often depict a seasonal behavior. In such cases, we can drastically reduce the number of stages by introducing a periodical analog of the Bellman equations, used in Markov decision process (MDP) and stochastic optimal control (SOC), and consequently applying a periodical variant of the SDDP algorithm. Since the computational complexity of the SDDP algorithm is explicitly related to the number of stages, the proposed periodical SDDP algorithm significantly reduces the computational time compared to the classical SDDP. This makes previously computationally intractable problems feasible to solve.

In the last part of the thesis (chapter 5), we explore various applications in energy, airline, and finance using the MSPPy package to illustrate the methodologies and algorithms we have developed. Further, we empirically study various numerical aspects of multistage stochastic programming.

CHAPTER 1

INTRODUCTION

We start with presenting the mathematical formulation of a multistage stochastic program. In multistage setting, the underlying data process is revealed sequentially. The data process is modeled as a stochastic process $(\xi_1, \ldots, \xi_T)$ so that ξ_1 is deterministic and $\xi_2, \ldots, \xi_T$ is to be revealed over time. We denote the history of the stochastic process up to time t by $\xi_{[t]} := (\xi_1, ..., \xi_t)$. The stochastic process $\{\xi_t\}$ is called stage-wise independent if ξ_t is independent of $\xi_{[t-1]}$ for $t = 2, \ldots, T$. The stochastic process $\{\xi_t\}$ is called Markovian if the conditional distribution of ξ_t given $\xi_{[t-1]}$ is the same as that of ξ_t given ξ_{t-1} for $t = 2, \ldots, T$. Sometimes to distinguish between these two types of stochastic processes, we denote stage-wise independent stochastic processes by $\{\xi_t\}$ and denote Markovian processes by $\{\eta_t\}$.

Decision variables in each stage are categorized into state variables and control variables, denoted by x and y respectively. At the beginning of stage t (giving $\xi_{[t-1]}$), the state variable x_{t-1} is regarded as information (also known as state) that flows from stage $t-1$ to the current stage. In the decision process, decision variables x_t, y_t $(t = 2, \ldots, T)$ are considered to be functions $\mathbf{x}_t(\xi_{[t]}, \eta_{[t]}), \mathbf{y}_t(\xi_{[t]}, \eta_{[t]})$ of the stochastic processes. The decision variables x_1, y_1 are deterministic and the initial value of state variable x_0 is supposed to be known. We call the sequence of decision variables $\pi := (x_1, y_1, \mathbf{x}_2, \mathbf{y}_2, \ldots, \mathbf{x}_T, \mathbf{y}_T)$ an implementable policy. We denote by Π the set of implementable policies satisfying feasibility constraints. This kind of notation is similar to the model used in the Optimal Control and it is beneficial to consider the state variables and control variables separately in implementations (see chapter 4). Nevertheless, for ease of notation, we sometimes consider (x, y) together as a decision vector and simply denote it by x.

For illustration purpose, we always deal with minimization problems in this thesis un-

less stated otherwise. The objective is then to a find an *implementable* policy that minimizes the expected cost as follows. Choosing expectation as our objective is natural due to the law of large numbers. Other choices will also be discussed.

$$\begin{aligned} &\min_{\pi\in\Pi}\mathbb{E}^{\pi}\left[\sum_{t=1}^{T}\gamma^{t-1}f_t(x_t,\xi_t)\right] \\ &\text{s.t. } x_t\in\mathscr{X}_t(x_{t-1},\xi_t), t=1,\dots,T \end{aligned} \tag{1.1}$$

Here $\gamma\in(0,1)$ is the discount factor, f_t are cost functions, and $\mathscr{X}_t$ are measurable closed valued functions. The expectation is taken with respect to policy $\pi\in\Pi$. The decision variables x_t are understood as x_t^{π}. It is assumed that the probability distribution of the data process $\xi_1,...,\xi_T$ does *not* depend on considered policies $\pi\in\Pi$. In other words, the underlying data process is considered as *exogenous*.

For every feasible implementable policy $\pi\in\Pi$ and a sample path $\xi_{[T]}$, we call $\sum_{t=1}^{T}\gamma^{t-1}f_t(x_t,\xi_t)$ a (realized) policy value. We call $\mathbb{E}^{\pi}\left[\sum_{t=1}^{T}\gamma^{t-1}f_t(x_t,\xi_t)\right]$ the expected policy value. The multistage stochastic program (1.1) is thus said to minimize the expected policy value over all feasible implementable policies.

Throughout the thesis, we focus on *linear* multistage stochastic programs, unless stated otherwise, in which the cost functions and the constraint functions are linear. The constraint functions $\mathscr{X}_t$ are written in the following two forms.

One formulation explicitly shows the so-called balance equation $B_tx_{t-1}+A_tx_t=b_t$ and shrinks other constraints to a closed set $\mathcal{X}_t$. This formulation is more clear when developing theory. We use it to demonstrate periodical SDDP in chapter 4,

$$\mathscr{X}_t(x_{t-1},\xi_t):=\{x_t: B_tx_{t-1}+A_tx_t=b_t, x_t\in\mathcal{X}_t\}$$

Another formulation combines all the constraints into matrix form and introduce additional control variables z_t acting as local copies of the past state variables x_{t-1}. As a

consequence, the balance constraint is simplified as $z_t = x_{t-1}$. As we will see this formulation is beneficial for numerical implementations and solving problems with integrality constraints. We will adopt it in chapter 2 and chapter 3.

$$\begin{aligned} f_t(x_t, \xi_t) =& u_t x_t \\ \mathscr{X}_t(x_{t-1}, \xi_t) :=& \{x_t : A_t x_t + B_t z_t \geq b_t, z_t = x_{t-1}\} \end{aligned}$$

By using tower property of conditional expectation and interchange of expectation and minimization, we have the equivalent nested formulation,

$$\begin{aligned} \min_{x_1 \in \mathscr{X}_1(x_0, \xi_1)} & f_1(x_1) + \gamma \mathbb{E}_{|\xi_1} \Big[\min_{x_2 \in \mathscr{X}_2(x_1, \xi_2)} f_2(x_2, \xi_2) \\ & + \cdots + \gamma \mathbb{E}_{|\xi_{[T-1]}} [\min_{x_T \in \mathscr{X}_T(x_{t-1}, \xi_T)} f_T(x_T, \xi_T)] \Big], \end{aligned} \tag{1.2}$$

The difficulty in solving formulation (1.1) or (1.2) is twofold. First, the computation of expectation often involves high-dimensional integral. Second, the computational complexity often grows exponentially with respect to the number of stages.

Throughout the thesis, we assume the following relatively complete recourse condition. This assumption can always be satisfied by adding slack variables and penalty costs.

Assumption 1 The multistage stochastic program has relatively complete recourse. That is, for almost every realizations of random variables ξ_t, and any states x_{t-1}, there always exists x_t such that $x_t \in \mathscr{X}_t(x_{t-1}, \xi_t)$ for $t = 1, \ldots, T$.

CHAPTER 2

METHODOLOGIES FOR SOLVING MULTISTAGE STOCHASTIC PROGRAMS

In this chapter we continue developing theory and demonstrate a standardized procedure we proposed in [1] to solve *linear* multistage stochastic programs (MSLP). As preparation to represent the software package in chapter 3, we will make the notation as explicit as possible, so that mathematical formulations and algorithms can be easily transformed into code. Throughout this chapter and chapter 3, we will explicitly unfold the decision variable x into the state variable x and the control (local) variable y and explicitly unfold the underlying data process ξ into stage-wise independent process ξ and Markov process η. Also as said in chapter 1, we will introduce an additional control variable z_t acting as local copies of the past state variables x_{t-1}.

2.1 Dynamic programming equations

We consider the following nested formulation, which suggests a way to write dynamic programming equations,

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0}} u_1^\top x_1 + v_1^\top y_1 + \gamma\mathbb{E}_{|\xi_1,\eta_1}\Big[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1}} u_2^\top x_2 + v_2^\top y_2$$
$$+\cdots+\gamma\mathbb{E}_{|\xi_{[T-1]},\eta_{[T-1]}}\big[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}}} u_T^\top x_T + v_T^\top y_T\big]\Big],$$

where x_t, y_t are the state variables and control variables respectively, z_t are the the control variables acting as copies of the past state, $u_t = u_t(\xi_t, \eta_t)$ and $v_t(\xi_t, \eta_t)$ are the corresponding costs, $b_t = b_t(\xi_t, \eta_t)$ are right side vectors, $A_t = A_t(\xi_t, \eta_t), B_t = B_t(\xi_t, \eta_t), C_t = C_t(\xi_t, \eta_t)$ are matrices of appropriate dimensions, and $\gamma \in (0, 1)$ is the discount factor.

2.1.1 Stage-wise independent problems

Stage-wise independent problems assume that the underlying data process is a stage-wise independent stochastic process, denoted by $\{\xi_t\}$. Then for $t = T, \ldots, 1$, the dynamic programming equations can be written as

$$Q_t(x_{t-1}, \xi_t) = \min_{\substack{A_t x_t + B_t z_t + C_t y_t \geq b_t \\ z_t = x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \gamma \mathcal{Q}_{t+1}(x_t), \tag{2.1}$$

where the expected cost-to-go (also called value, recourse) functions,

$$\mathcal{Q}_{t+1}(x_t) := \begin{cases} \mathbb{E}\{Q_{t+1}(x_t, \xi_{t+1})\}, \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T, \end{cases} \tag{2.2}$$

do not depend on the underlying stochastic process. Note that if ξ_{t+1} has a discrete distribution with a finite number of possible realizations, the corresponding expectation in (2.2) can be written in a form of finite summation.

2.1.2 Markovian problems

Markovian problems assume that the underlying data process is a Markov process, denoted by $\{\eta_t\}$, or more general, a combination of a Markov process $\{\eta_t\}$ and a stage-wise independent process $\{\xi_t\}$ (independent from $\{\eta_t\}$). For $t = T, \ldots, 1$, the respective dynamic programming equations are

$$Q_t(x_{t-1}, \xi_t, \eta_t) = \min_{\substack{A_t x_t + B_t z_t + C_t y_t \geq b_t \\ z_t = x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \gamma \mathcal{Q}_{t+1}(x_t, \eta_t), \tag{2.3}$$

where the expected cost-to-go functions,

$$\mathcal{Q}_{t+1}(x_t, \eta_t) := \begin{cases} \mathbb{E}_{|\eta_t}\{Q_{t+1}(x_t, \xi_{t+1}, \eta_{t+1})\}, \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T, \end{cases} \tag{2.4}$$

depend on η_t rather than the whole history $\eta_{[t-1]}$ of the Markov process. If the process $\{\eta_t\}$ follows a (non-homogeneous) Markov chain with Markov state space $\{s_t | s_t \in S_t\}$, $t = 1, 2 \ldots, T$ (S_1 is a singleton) and transition matrix $\left\{\mathbf{P}^t_{s_{t-1}, s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\right\}$, $t = 2, \ldots, T$, the expected cost-to-go functions in (2.4) can be written as,

$$\mathcal{Q}_{t+1}(x_t, s_t) = \sum_{s_{t+1} \in S_{t+1}} \mathbf{P}^t_{s_t, s_{t+1}} \mathbb{E}\{Q_{t+1}(x_t, \xi_{t+1}, s_{t+1})\},$$

where $s_t \in S_t, t = T-1, \ldots, 1$. Further, if ξ_{t+1} is discrete with a finite number of possible realizations, the conditional expectation can be written as a finite sum.

We end this section with a remark that the relatively complete recourse condition guarantees that $Q_t(x, \xi_t)$ ($Q_t(x, \xi_t, \eta_t)$) is finite for any x and realizations of the random data vectors. Moreover, since $Q_t(\cdot, \xi_t)$ ($Q_t(\cdot, \xi_t, \eta_t)$) is convex, it is subdifferentiable everywhere and its subgradients, at a point x, are given by optimal solutions of the dual problem corresponding to equation $z_t = x$.

2.2 Discretization

Stage-wise independent stochastic processes $\{\xi_t\}$ and Markov processes $\{\eta_t\}$ may have large/infinite number of possible realizations. Consequently, these processes should be discretized in order to make the problem solvable. Before we discuss certain discretization techniques, we introduce the following two terms.

The original problem (2.1) or (2.3) is referred to as the *true problem*. The true problem is *continuous* if the underlying stochastic process follows a continuous distribution. The true problem is *discrete* if the underlying stochastic process follows a discrete distribution.

It is further categorized as a *finite (infinite) discrete* true problem if the discrete distribution has finite (infinite) possible values. Continuous and infinite discrete true problems are not amenable for a finite representation in a computer and should be discretized. A finite discrete true problem may be computationally intractable when the discrete distribution has a large number of possible values.

A discretization of the true problem is called a *discretized problem*. As we will discuss in this section, stage-wise independent processes can be discretized by the Sample Average Approximation (SAA) and Markov processes can be discretized by the SAA through a Time Series approach (TS) or by Markov chain approximation (MCA). We refer to the discretized problem as an SAA discretized problem or a Markov chain discretized problem correspondingly.

We would like to mention that though the discretized problem could be solvable, our real interest is still in the true problem rather than the discretized problem. Without careful analysis, an optimal policy generated by the discretized problem does not give a guarantee about its performance for the true problem. It may be even not feasible to implement for the true problem.

2.2.1 Sample Average Approximation (SAA) approach

Sample average approximation (SAA) generates independent identically distributed (i.i.d.) samples from the true distribution and uses sample average to approximate the expectation. For the state-wise independent stochastic process $\{\xi_t\}$, it is natural to discretize marginal distributions of random vectors ξ_t. Paper [2] shows that under mild regularity condition, the optimal value and an optimal solution of the SAA problem converge with probability one to the true problem. It also provides estimate on the sample size needed to obtain an ϵ-optimal solution. That is, to solve the true problem with accuracy $\varepsilon > 0$ by solving the SAA problem with accuracy $\delta \in [0, \varepsilon)$, for a given confidence one needs a sample size N of order $O\left((\varepsilon - \delta)^{-2}\right)$. One important point here is that the required sample size grows

linearly with the dimension of the decision variables. Therefore, in some cases it is able to deal with the first difficulty introduced in chapter 1.

2.2.2 Time Series (TS) approach

Sometimes the Markovian data process $\{\eta_t\}$ can be modeled using the following time series equations (higher order time series models can be transformed to first order by adding auxiliary variables),

$$\eta_t = \mu_t + \Phi_t \eta_{t-1} + a_t \text{ (VAR)}, \tag{2.5}$$

$$\eta_t = (\mu_t + \Phi_t \eta_{t-1}) a_t \text{ (VAR*)}, \tag{2.6}$$

$$\ln \eta_t = \mu_t + \Phi_t \ln \eta_{t-1} + a_t \text{ (VGAR)}, \tag{2.7}$$

where μ_t, a_t are m dimensional vectors, Φ_t is an $m \times m$ coefficient matrix and a_t is a white noise vector process with zero mean in (2.5) and (2.7) and mean 1 in (2.6); the multiplication in (2.6) is understood pointwise.

If η_t appears on the right hand side of constraints and follows VAR/VAR* models, one can add η_t as additional state variables to the true problem. The reformulated true problem, with a_t viewed as the underlying random data process, is stage-wise independent and thus can be discretized by SAA. On the other hand, if η_t follows VGAR model, or it appears in the objective or left hand side of constraints, adding η_t as additional state variables will destroy the linearity of the true problem. Cases like that should resort to Markov chain approximation instead.

A recent development in [3] is to consider the dual of the multistage stochastic program. The uncertainties in objective for the primal problem are then transformed into the uncertainties on right hand sides for the dual problem. Therefore, we can implement the TS approach on the dual program instead.

2.2.3 Markov chain approximation (MCA)

Another more general approach to discretize the Markov process $\{\eta_t\}$ is by Markov chain approximation (MCA). Suppose the sample space of the $\{\eta_t\}$ at stage t is Ω_t (η_1 is known and Ω_1 is a singleton). Suppose further that F is the joint distribution of $\eta_1 \ldots, \eta_T$. From stage two on, we are going to partition Ω_t into K_t subsets $\{\omega_{tk}, k = 1, \ldots, K_t\}$ centered at μ_{tk} and let the centers μ_{tk} to be the Markov states. Such partition is known as the *Voronoi cells*. Let U_t denote the partition $\{\omega_{tk}, k = 1, \ldots, K_t\}$ at stage t and U denote the whole partition $\{U_t, t = 1, \ldots, T\}$. Our objective is to search for a partition U that minimizes the Euclidean distance from the centers (Markov states) and the Markov process $\{\eta_t\}$,

$$\min_U \ \mathbb{E}_\eta \left\{ \sum_{t=2}^{T} \min_{k=1,\ldots,K_t} \|\eta_t - \mu_{tk}\|_2^2 \right\}. \tag{2.8}$$

Three methods can be used to solve the above problem.

The first approach is by virtue of the *Sample Average Approximation* (SAA). We draw S independent sample paths $\{\hat{\eta}_1^s, \ldots, \hat{\eta}_T^s\}, s = 1, \ldots, S$ from F and construct an SAA of (2.8),

$$\min_U \sum_{s=1}^{S} \sum_{t=2}^{T} \min_{k=1,\ldots,K_t} \|\hat{\eta}_t^s - \mu_{tk}\|_2^2,$$

which is equivalent to,

$$\min_U \sum_{t=2}^{T} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat{\eta}_t^s - \mu_{tk}\|_2^2 = \sum_{t=2}^{T} \min_{U_t} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat{\eta}_t^s - \mu_{tk}\|_2^2.$$

The summands $\min_{U_t} \sum_{s=1}^{S} \min_{k=1,\ldots,K_t} \|\hat{\eta}_t^s - \mu_{tk}\|_2^2$ are identical to the objective of a K-means problem. We can then use the Lloyd's algorithm [4] to find a local optimum.

Another approach is using the *Stochastic Approximation* (SA) algorithm. Initialize μ_{tk} as μ_{tk}^0, $k = 1, \ldots, K_t$. With a sequence of stepsizes $(\beta_s)_{s=1}^S$, the Markov states are updated

recursively by stochastic approximation,

$$\mu_{ti}^{s} = \begin{cases} \mu_{ti}^{s-1} + \beta_s(\hat{\eta}_t^s - \mu_{ti}^{s-1}), \text{ if } i = \text{argmin}_{k\in\{1,\ldots,K_t\}}\{\|\hat{\eta}_t^s - \mu_{tk}^{s-1}\|_2^2\} \\ \mu_{ti}^{s-1}, \text{ otherwise.} \end{cases}$$

We can for example, set a diminishing step size sequence $\beta_s = 1/s, s = 1, \ldots, S$, to ensure that it converges to a local minimum [5].

The third approach is utilizing the *Robust Stochastic Approximation* (RSA) approach, originally developed for convex stochastic problems [6]. Given a specified number of iterations N, a constant step size $\frac{c}{\sqrt{N}}$ is used (c is a constant) and the final approximating solution is the average of the iterates. This algorithm has $O(N^{-\frac{1}{2}})$ rate of convergence. In our case, the objective function is not convex and thus the convergence is not guaranteed. Nevertheless, it appears to work well in practice. We attach pseudo code for this approach in Algorithm 1.

Algorithm 1 Markov chain approximation (MCA) through Robust Stochastic Approximation (RSA)

1: Given a sample path generator F, intended number of Markov states K_t, $t = 1, \ldots, T(K_1 = 1)$, intended number of training sample paths N.
2: Randomly initialize $\mu_{tk}^0, k \in \{1, \ldots, K_t\}, t = 2, \ldots, T$
3: **for** $s = 1, \ldots, S$ **do**
4: Draw one sample path $\{\hat{\eta}_1, \hat{\eta}_2, \ldots, \hat{\eta}_T\}$ from F
5: **for** $t = 2, \ldots, T$ **do**
6: $m = \text{argmin}_k\{\|\hat{\eta}_t - \mu_{tk}^{s-1}\|_2^2, k = 1, \ldots, K_t\}$
7: Update $\mu_{ti}^s = \begin{cases} \mu_{ti}^{s-1} + \frac{1}{\sqrt{N}}(\hat{\eta}_t - \mu_{ti}^{s-1}), \text{ if } i = m \\ \mu_{ti}^{s-1}, \text{ if } i \in \{1, \ldots, K_t\} \setminus m \end{cases}$
8: **end for**
9: **end for**
10: Set $\mu_{tk} = \frac{1}{S}\sum_{s=1}^{S} \mu_{tk}^s, k = 1, \ldots, K_t, t = 2, \ldots, T$.

Finally, the transition matrix is approximated by relative frequencies,

$$\mathbf{P}_{ij}^t = \frac{\sum_{s=1}^{S} 1_{\hat{\eta}_{t-1}^s \in \omega_{t-1,i}} 1_{\hat{\eta}_t^s \in \omega_{tj}}}{\sum_{s=1}^{S} 1_{\hat{\eta}_{t-1}^s \in \omega_{t-1,i}}}, t = 2, \ldots, T.$$

Where $1_{(\cdot)}$ is the indicator function. There is no clear evidence showing that one approach is superior to the others. We will empirically compare the three methods in section 5.1.5.

As above, for $t = 2, \ldots, T$, the SAA discretizes ξ_t by generating i.i.d. samples $\xi_t^j, j = 1, \ldots, N_t$, and MCA discretizes $\{\eta_t\}_{t=1}^T$ by a Markov chain with Markov state spaces $\{s_t | s_t \in S_t\}$, $t = 1, \ldots, T$, and transition matrices $\{\mathbf{P}^t_{s_{t-1}, s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2, \ldots, T$. The SAA discretization of the true problem (2.7) takes the form of,

$$\begin{aligned} \tilde{Q}_t(x_{t-1}, \xi_t^j) &= \min_{\substack{A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj} \\ z_t = x_{t-1}}} u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \tilde{\mathcal{Q}}_{t+1}(x_t), \\ \tilde{Q}_1(x_0, \xi_1) &= \min_{\substack{A_1 x_1 + B_1 z_1 + C_1 y_1 \geq b_1 \\ z_1 = x_0}} u_1^\top x_1 + v_1^\top y_1 + \gamma \tilde{\mathcal{Q}}_2(x_1), \end{aligned} \tag{2.9}$$

where $j = 1, \ldots, N_t$, $t = T, \ldots, 2$, and the approximate expected cost-to-go is defined as

$$\tilde{\mathcal{Q}}_{t+1}(x_t) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{j=1}^{J_{t+1}} \tilde{Q}_{t+1}(x_t, \xi_{t+1}^j), \text{ for } t \neq T, \\ 0, \text{ for } t = T, \end{cases}$$

The Markov chain discretization of the true problem (2.3) takes the form of

$$\begin{aligned} \tilde{Q}_t(x_{t-1}, \xi_t^j, s_t) &= \min_{\substack{A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj} \\ z_t = x_{t-1}}} u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \tilde{\mathcal{Q}}_{t+1}(x_t, s_t), \\ \tilde{Q}_1(x_0, \xi_1, s_1) &= \min_{\substack{A_1 x_1 + B_1 z_1 + C_1 y_1 \geq b_1 \\ z_1 = x_0}} u_1^\top x_1 + v_1^\top y_1 + \gamma \tilde{\mathcal{Q}}_2(x_1, s_1), \end{aligned} \tag{2.10}$$

where $j = 1, \ldots, N_t, s_t \in S_t$, $t = T, \ldots, 2$, and the approximate expected cost-to-go is defined as

$$\tilde{\mathcal{Q}}_{t+1}(x_t, s_t) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{s_{t+1} \in S_{t+1}} \sum_{j=1}^{J_{t+1}} \mathbf{P}^t_{s_t, s_{t+1}} \tilde{Q}_{t+1}(x_t, \xi_{t+1}^j, s_{t+1}), \text{ for } t \neq T, \\ 0, \text{ for } t = T. \end{cases}$$

2.3 Extensive formulation of the discretized problem

We can formulate the discretized problem equivalently as a deterministic linear program. We refer to such an equivalent deterministic linear program as *Extensive Formulation*. Let $\{n|n \in \tau\}$ be the set of nodes in a constructed scenario tree. Each node n at stage t corresponds to a specific realization of data process $(u_n, v_n, A_n, B_n, C_n, b_n)$ at stage t and also corresponds to a sample path up to stage t. We denote the ancestor of node n by $a(n)$. We denote the probability of a sample path going through node n by P_n. The discount factor at node n is denoted by γ_n, which is equal to γ^{t-1} if the node n is at stage t. For each $n \in \tau$, we associate a single decision vector (x_n, y_n). In this way, the nonanticipativity constraints automatically satisfy. The extensive formulation then takes the form of,

$$\min \sum_{n \in \tau} \gamma_n P_n (u_n^\top x_n + v_n^\top y_n)$$
$$\text{s.t. } A_n x_n + B_n y_n + C_n x_{a(n)} \geq b_n.$$

Solving the extensive formulation provides us with an implementable policy only for the considered scenario tree. As it was mentioned before, the computed optimal policy does not provide an implementable policy for the true problem, at least not in a direct way. Moreover, the number of required scenarios in a constructed scenario tree grows exponentially with the increase of the number of stages. Nevertheless, this approach could be applicable when the number of stages is small (say not larger than 3) and could be useful for verification of a considered model.

The scenario tree can be either stage-wise independent or Markovian. The computation of the probability P_n is straightforward as follows. Firstly, suppose the discretized problem is SAA. For each stage t ($t = 1, \ldots, T$), denote stage-wise independent scenarios by $\{j_t|j_t \in 1, \ldots, N_t\}$ ($J_1 = 1$) and related probability measures by $\{P_{j_t}^t|j_t \in 1, \ldots, N_t\}$, $t = 1, \ldots, T$ ($P_{j_1}^1 = 1$). Consider node n at stage t and its corresponding sample path

$(j_1, j_2, \ldots, j_t)$. Then P_n is given by

$$P_n = P^1_{j_1} P^2_{j_2} \ldots P^t_{j_t}.$$

Secondly, suppose the discretized problem is Markov chain type and has only the Markov chain stochastic process. Let the Markov state space at stage t be S_t (initial state space S_1 is a singleton) and the transition matrix between stage $t-1$ and stage t be $\{\mathbf{P}^t_{s_{t-1},s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2, \ldots, T$. Consider node n at stage t and its corresponding sample path $(s_1, s_2, \ldots, s_t)$. Then P_n is given by

$$P_n = \mathbf{P}^2_{s_1,s_2} \mathbf{P}^3_{s_2,s_3} \ldots \mathbf{P}^t_{s_{t-1},s_t}.$$

Finally, suppose that the Markov chain discretized problem also has the stage-wise independent stochastic process (independent of the Markov chain process). Then we can combine the two corresponding scenario trees into one single scenario tree with P_n given by

$$P_n = P^1_{j_1} P^2_{j_2} \ldots P^t_{j_t} \mathbf{P}^2_{s_1,s_2} \mathbf{P}^3_{s_2,s_3} \ldots \mathbf{P}^t_{s_{t-1},s_t}.$$

2.4 Solving the discretized problem

2.4.1 Algorithm based on the extensive formulation

The extensive solver uses the extensive formulation and solves a single deterministic, although may be large, linear program. As it was already mentioned, the number of variables and constraints in the extensive formulation grows exponentially with increase of the number of stages. So it suffers from exponential complexity and is only able to solve relatively small problems. In addition, as mentioned above, the solution provided by the extensive formulation is only implementable for the discretized problem. It is unclear what such solution means to the true problem. Nevertheless, it could be useful for small scale problems

and could be used as a validation tool for the correctness of other algorithms.

2.4.2 SDDP algorithm

The SDDP algorithm [7] leverages convexity of the value functions and appears to enjoy more or less linear complexity with respect to the number of stages (this is an empirical observation; theoretical evidence can be found in [8], [9]). So it somehow mitigates the second difficulty introduced in chapter 1. Furthermore, the SDDP algorithm provides feasible implementable policies not only for the discretized problem, but also for the true problem. This is beneficial since solving the true problem is of our primal interest.

Stage-wise independent SDDP

The SAA discretized problem (2.9) is solved by the classical SDDP algorithm as shown in Algorithm 2.

The computed approximations $\mathfrak{Q}_2(\cdot), \ldots, \mathfrak{Q}_T(\cdot)$ of the value functions and a feasible first stage solution $\bar{x}_1$ generates a feasible implementable policy $x_t(\xi_{[t]})$ for the discretized as well as the true problem as follows. For any realization (sample path) of the process $\{u_t, v_t, A_t, B_t, C_t, b_t\}$, $t = 2, \ldots, T$, we associate a decision vector $x_1(= \bar{x}_1), x_2, \ldots, x_T$, generated by solving recursively

$$\min \{u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) : A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = x_{t-1}\}, t = 2, \ldots, T.$$

By the assumption of relatively complete recourse, it generates a feasible implementable policy for the considered problem. If we restrict sample paths $\xi_{[t]}$ to the discretized process, the implied policy $x_t(\xi_{[t]})$ is feasible and implementable for the discretized problem. On the other hand, if we consider the sample paths $\xi_{[t]}$ generated from the distribution of the true problem, the policy $x_t(\xi_{[t]})$ is feasible and implementable for the true problem. As a result, the expected value $\mathbb{E}\left\{\sum_{t=1}^{T}\left[\gamma^{t-1}(u_t^\top x_t + v_t^\top y_t)\right]\right\}$ gives an upper bound for the

Algorithm 2 Stage-wise independent SDDP

Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}, b_{tj}\}_{1\le j\le N_t}, t = 2, \ldots, T$
Given initial value $\bar{x}_0$
Given initial approximation of value functions : $\mathfrak{Q}_t^0(\cdot) = \{\theta : \theta(\cdot) \ge l_t\}, t = 2, \ldots, T$
Initialize: i = 1, LB = -∞
while no stopping criterion is met **do**
(Forward Step)
  for $t = 1, \ldots, T$ **do**
    if $t > 1$ **then** draw a sample $(u_t, v_t, A_t, B_t, C_t, b_t)$ from Ω_t
    end if
    $\bar{x}_t, \bar{y}_t =$ argmin $\left\{u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}^{i-1}(x_t) : A_t x_t + B_t z_t + C_t y_t \ge b_t, z_t = \bar{x}_{t-1}\right\}$
  end for
(Backward Step)
  for $t = T, \ldots, 2$ **do**
    for $j = 1, \ldots, N_t$ **do**
      Solve min $\left\{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}^i(x_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge b_{tj}, z_t = \bar{x}_{t-1}\right\}$ and get the optimal value $\mathscr{V}_{tj}$ and a dual solution $\mathscr{G}_{tj}$ corresponding to $z_t = \bar{x}_{t-1}$
    end for
    $\mathscr{V}_t := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}, \mathscr{G}_t := \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj}$
    $\mathfrak{Q}_t^i \leftarrow \left\{\theta \in \mathfrak{Q}_t^{i-1}, \theta(x) \ge \mathscr{G}_t(x - \bar{x}_{t-1}) + \mathscr{V}_t\right\}$
  end for
  Policy value $= \sum_{t=1}^{T} \left[\gamma^{t-1}(u_t^\top \bar{x}_t + v_t^\top \bar{y}_t)\right]$
  LB $= \min\left\{u_1^\top x_1 + v_1^\top y_1 + \gamma \mathfrak{Q}_2^i(x_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \ge b_1, z_1 = \bar{x}_0\right\}$
  $i = i + 1$
end while

optimal value of the true or the discretized problem depending on what sample paths are considered.

Markov chain SDDP

The Markov chain discretized problem (2.10) can be solved by a Markov chain version of SDDP algorithm. For instance, Algorithm 3 solves a minimization problem in which $\{u_t\}, \{v_t\}, \{A_t\}, \{B_t\}, \{C_t\}$ are stage-wise independent stochastic processes and $\{b_t\}$ is a Markov chain process,

The computed approximations $\mathfrak{Q}_2(\cdot, s_1), \ldots, \mathfrak{Q}_T(\cdot, s_{T-1})$, $s_1 \in S_1, \ldots, s_{T-1} \in S_{T-1}$ and a feasible first stage solution $\bar{x}_1$ can generate a feasible implementable policy $x_t(\xi_{[t]}, \eta_{[t]})$ for the true problem as follows. For any realization (sample path) of the process

Algorithm 3 Markov chain SDDP

Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}\}_{1\le j\le N_t}$, state space of Markov chain S_t, transition matrix $\{\mathbf{P}_{s_{t-1},s_t} | s_{t-1} \in S_{t-1}, s_t \in S_t\}$, $t = 2, \ldots, T$, initial state s_1
Given initial approximation of the value functions:
$\mathfrak{Q}_t^0(\cdot, s_{t-1}) = \{\theta(\cdot, s_{t-1}) | \theta(\cdot, s_{t-1}) \ge L_0\}, \forall s_{t-1}$
Initialize: i = 1, LB = -∞
while no stopping criterion is met **do**
(Forward Step)
  for $t = 1, \ldots, T$ **do**
    if $t > 1$ **then** draw a sample $(u_t, v_t, A_t, B_t, C_t)$ from Ω_t; draw a state s_t w.p. $\mathbf{P}_{s_{t-1},s_t}$
    end if
    $\bar{x}_t, \bar{y}_t = \text{argmin} \left\{u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}^{i-1}(x_t, s_t) : A_t x_t + B_t z_t + C_t y_t \ge s_t, z_t = \bar{x}_{t-1}\right\}$
  end for
(Backward Step)
  for $t = T, \ldots, 2$ **do**
    for $j = 1, \ldots, N_t$ **do**
      for $s_t \in S_t$ **do**
        Solve $\min \left\{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}^i(x_t, s_t) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \ge s_t, z_t = \bar{x}_{t-1}\right\}$ and get the optimal value $\mathscr{V}_{tj,s_t}$ and a dual solution $\mathscr{G}_{tj,s_t}$ corresponding to $z_t = \bar{x}_{t-1}$
      end for
    end for
    for $s_{t-1} \in S_{t-1}$ **do**
      $\mathscr{V}_{t,s_{t-1}} := \frac{1}{N_t} \sum_{j=1}^{N_t} \sum_{s_t \in S_t} [\mathbf{P}_{s_{t-1},s_t} \mathscr{V}_{tj,s_t}]$,
      $\mathscr{G}_{t,s_{t-1}} := \frac{1}{N_t} \sum_{j=1}^{N_t} \sum_{s_t \in S_t} [\mathbf{P}_{s_{t-1},s_t} \mathscr{G}_{tj,s_t}]$
      $\mathfrak{Q}_t^i(\cdot, s_{t-1}) \leftarrow \left\{\theta \in \mathfrak{Q}_t^{i-1}(\cdot, s_{t-1}), \theta(x, s_{t-1}) \ge \mathscr{G}_{t,s_{t-1}}(x - \bar{x}_{t-1}) + \mathscr{V}_{t,s_{t-1}}\right\}$
    end for
  end for
  Policy value $= \sum_{t=1}^{T} \left[\gamma^{t-1}(u_t^\top \bar{x}_t + v_t^\top \bar{y}_t)\right]$
  LB $= \min \left\{u_1^\top x_1 + v_1^\top y_1 + \gamma \mathfrak{Q}_2(x_1, s_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \ge s_1, z_1 = x_0\right\}$
  $i = i + 1$
end while

$\{u_t, v_t, A_t, B_t, C_t, b_t\}$, $t = 2, \ldots, T$, assume s_t^* is the closest Markov state to b_t, i.e., $s_t^* = \text{argmin}_{s_t} \{\|s_t - b_t\|_2^2, s_t \in S_t\}$. We then associate a decision vector $x_1 (= \bar{x}_1), x_2, \ldots, x_T$, generated by solving recursively $\min \{u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t, s_t^*) : A_t x_t + B_t z_t + C_t y_t \ge s_t^*, z_t = x_{t-1}\}, t = 2, \ldots, T$. By Assumption 1, it is a feasible implementable policy for the true problem. Again, the expected policy value $\mathbb{E}\left\{\sum_{t=1}^{T} \left[\gamma^{t-1}(u_t^\top x_t + v_t^\top y_t)\right]\right\}$ gives

an upper bound for the true or discretized problem depending on what sample paths are considered.

2.4.3 Stopping criteria

Three kinds of stopping criteria can be considered for the SDDP algorithm. The first criterion is an upper limit of certain characteristics including time, the number of iterations and the number of iterations that the lower bound doesn't change. The algorithm will be stopped if one of the upper limits is reached.

The second criterion is based on estimation of the optimality gap [10]. It is estimated by evaluating the value of the constructed policy on the discretized problem employing Monte Carlo simulation. One can specify the frequency of such evaluations and the number of simulations M. In each simulation, a sample path $(\xi_{[t]}^m, \eta_{[t]}^m)$ is generated independently from the discretized problem, leading to an optimal solution $\bar{x}_{tm}(\xi_{[t]}^m, \eta_{[t]}^m)$, $\bar{y}_{tm}(\xi_{[t]}^m, \eta_{[t]}^m)$ and policy values

$$V_m = \sum_{t=1}^{T} \left[\gamma^{t-1}(u_{tm}^T \bar{x}_{tm} + v_{tm}^T \bar{y}_{tm})\right],\ m = 1, \ldots, M.$$

Define sample mean and sample variance as

$$\bar{V}_M := M^{-1} \sum_{m=1}^{M} V_m$$

$$S_M^2 := \frac{1}{M-1} \sum_{m=1}^{M} (V_m - \bar{V}_M)^2$$

Let v^* be the expected value of the computed policy. By the CLT we have that $\frac{\bar{V}_M - v^*}{S_M/\sqrt{M}}$ converges in distribution to standard normal $\mathcal{N}(0,1)$. It follows that the approximate $(1-\alpha)$ one-sided confidence interval for the expected policy values is $\bar{V}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}}$, where $\mathbb{P}(Z \geq z_{1-\alpha}) = \alpha$ with $Z \sim \mathcal{N}(0,1)$. The upper end of the CI provides an upper bound of the optimal value of the discretized problem with confidence of approximately $1-\alpha$. Hence,

with confidence $1-\alpha$, the optimality gap for the discretized problem is smaller than

$$gap = \frac{\bar{V}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}} - \text{LB}}{\text{LB}}, \tag{2.11}$$

where LB is the lower bound obtained from the backward step. One can specify an intended tolerance ε for the optimality gap to stop the algorithm. If *gap* $\leq \varepsilon$, then one can be $(1-\alpha)$ confident that optimality gap for the discretized problem is not larger than ε. Note that the estimate (2.11) of the optimality gap is somewhat conservative. Its conservativeness depends on how good is the lower bound estimate and how large is the standard deviation $S_M/\sqrt{M}$.

The third criterion is based on stabilization of the expected policy value. If additional iterations provide little improvement of the expected policy value, it is reasonable to stop the algorithm. We compare a recent policy $\hat{x}_t$ with a past policy $\bar{x}_t$ by independently generating M sample paths. For each generated sample path $(\xi^m_{[t]}, \eta^m_{[t]})$, we obtain an optimal solution $\bar{x}_t(\xi^m_{[t]}, \eta^m_{[t]}), \bar{y}_t(\xi^m_{[t]}, \eta^m_{[t]})$ and $\hat{x}_t(\xi^m_{[t]}, \eta^m_{[t]}), \hat{y}_t(\xi^m_{[t]}, \eta^m_{[t]})$. Consequently the differences of policy values

$$D_m := \sum_{t=1}^{T} \left[\gamma^{t-1}(u_t^\top \bar{x}_t + v_t^\top \bar{y}_t - u_t^\top \hat{x}_t - v_t^\top \hat{y}_t)\right],\ m = 1, ..., M,$$

are computed.

Note that D_m is an unbiased estimator of the difference D of the two expected policy values. Similarly as above, the approximate $(1-\alpha)$ one-sided confidence interval for the difference D is given by $\bar{D}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}}$, where $\bar{D}_M$ is the sample average and S^2_M is the sample variance of $D_1, ..., D_M$. One can specify an intended threshold of difference ϵ_D to stop the algorithm. If $\bar{D}_M + z_{1-\alpha}\frac{S_M}{\sqrt{M}} \leq \epsilon_D$, then one can be $(1-\alpha)$ confident that improvement of the expected policy value is not larger than ϵ_D.

2.4.4 Regularization

The regularized SDDP algorithm adds a penalty term in the objective to reduce oscillating around feasible regions. The idea of quadratically regularizing the decomposition algorithm for two stage problems can be found in [11]. Such an idea can not be generalized directly to the multistage setting since we need to look at all incumbent solutions corresponding to all scenarios, which are often astronomically many. We present here an approach to regularization in the multistage setting proposed in [12]. The essential idea is to consider a single incumbent solution that can be shared among scenarios.

Recall that in each forward step in iteration i of the classical SDDP, we obtain trial solution $\bar{x}_t, \bar{y}_t$ by solving,

$$\min u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}^i(x_t) : A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = \bar{x}_{t-1}$$

The current approximation $\mathfrak{Q}_{t+1}^i(\cdot)$ to the value function may be weak and hence solving the above problem can lead us to an unfavorable region. Regularization mitigates the inaccuracy of current approximation by making the solution not too far away from the previous one. Suppose the trial solutions obtained from iteration $i-1$ is $\bar{x}_t^{i-1}$. Then for iteration i, the trial solutions are obtained by solving,

$$\begin{aligned} \min \ & u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}^i(x_t) + \frac{r^{k-1}}{2}(B_t\bar{x}_t - B_t\bar{x}_t^{i-1})^\top S_t (B_t\bar{x}_t - B_t\bar{x}_t^{i-1}) \\ & A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = \bar{x}_{t-1}, \end{aligned}$$

where the decay factor $r \in (0,1)$ and S_t is a positive semi-definite matrix to properly scale the penalty term. A popular choice of S_t is ωI, where I is the identity matrix and ω is the weight factor. The seemingly more natural regularization is to penalize the jump of x_t. But as shown in [12], convergence of the regularized SDDP algorithm can only be established if penalizing the jump of $B_t x_t$. Partial reason for that is because the recourse $\mathcal{Q}_{t+1}^i$ is actually

depending only on $B_t x_t$ (in [12], this is called post-decision state and the value functions is written as functions of the post-decision state).

There is no theoretical guarantee that the above quadratic regularization increases the convergence rate of the SDDP algorithm. In sections 5.1.8 and 5.1.10, we will numerically investigate the effect of regularization.

2.4.5 Parallelization

In this section, we will see how the SDDP algorithm can be parallelized in the stage-wise independent setting. Parallelizations in other cases are similar. As shown in Algorithm 4, it does M forward steps and backward steps for each iteration. These M jobs are independent and thus can be allocated to multiple processes.

It is worth noting that this approach has a subtle difference from the serial SDDP. In the serial SDDP, cuts are added for every single forward and backward step. While in the parallel SDDP, cuts are added every M forward and backward steps. Thus given the same number of forward and backward steps, the parallel SDDP will be less accurate than the serial SDDP.

It is also possible to parallelize the computation of cutting planes in the backward pass. But it loses the benefit of warm starts in solving linear programs by the dual simplex algorithm. For this reason this kind of parallelization is not adopted in this thesis.

2.5 Rolling horizon based algorithm

In sections 2.3 and 2.4.1, we introduce an algorithm based on the extensive formulation and discuss its two shortcomings. That is, it does not provide implementable policies for the true problem and it suffers from exponential complexity with respect to the number of stages. In this section, we develop a variant of this algorithm that deals with these shortcomings.

For any realization (sample path) of a general (stage-wise independent, Markovian, or

Algorithm 4 Parallel stage-wise independent SDDP

Given discretization $\Omega_t = \{u_{tj}, v_{tj}, A_{tj}, B_{tj}, C_{tj}, b_{tj}\}_{1\leq j\leq N_t}, t = 2, \ldots, T$
Given initial approximation of value functions : $\mathfrak{Q}_t^0(\cdot) = \{\theta : \theta(\cdot) \geq l_t\}, t = 2, \ldots, T$
Initialize: i = 1, LB = -∞
while no stopping criterion is met **do**
  for $m = 1, \ldots, M$ **do**
  (Forward Step)
    for $t = 1, \ldots, T$ **do**
      if $t > 1$ **then** draw a sample $(u_t, v_t, A_t, B_t, C_t, b_t)$ from Ω_t
      end if
      $\bar{x}_t^m, \bar{y}_t^m = \text{argmin}\left\{u_t^\top x_t + v_t^\top y_t + \gamma\mathfrak{Q}_{t+1}^{i-1}(x_t) : A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = \bar{x}_{t-1}^m\right\}$
    end for
    $V_m = \sum_{t=1}^T \left[\gamma^{t-1}(u_t^\top x_t^m + v_t^\top y_t^m)\right]$
  (Backward Step)
    for $t = T, \ldots, 2$ **do**
      for $j = 1, \ldots, N_t$ **do**
        Solve $\min\left\{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma\mathfrak{Q}_{t+1}^i(x_t) : A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj}, z_t = \bar{x}_{t-1}\right\}$ and get the optimal value $\mathscr{V}_{tj}^m$ and a dual solution $\mathscr{G}_{tj}^m$ corresponding to $z_t = \bar{x}_{t-1}$
      end for
      $\mathscr{V}_t^m := \frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{V}_{tj}^m, \mathscr{G}_t^m := \frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{G}_{tj}^m$
    end for
  end for
  $\mathfrak{Q}_t^i \leftarrow \{\theta \in \mathfrak{Q}_t^{i-1}, \theta(x_{t-1}) \geq \mathscr{G}_t^m(x_{t-1} - \bar{x}_{t-1}^m) + \mathscr{V}_t^m, m = 1, 2 \ldots, M\}$
  LB $= \min\{u_1^\top x_1 + v_1^\top y_1 + \gamma\mathfrak{Q}_2^i(x_1) : A_1x_1 + B_1z_1 + C_1y_1 \geq b_1, z_1 = x_0\}$
  sample mean $\bar{V} = (\sum_{i=1}^M V_i)/M$, sample standard deviation $\sigma(V) = \sqrt{\frac{\sum_{i=1}^M (V_i - \bar{V})^2}{M-1}}$
  CI of the expected policy value: $[\bar{V} - z_{1-\alpha/2}\frac{\sigma(V)}{\sqrt{M}}, \bar{V} + z_{1-\alpha/2}\frac{\sigma(V)}{\sqrt{M}}]$
  $i = i + 1$
end while

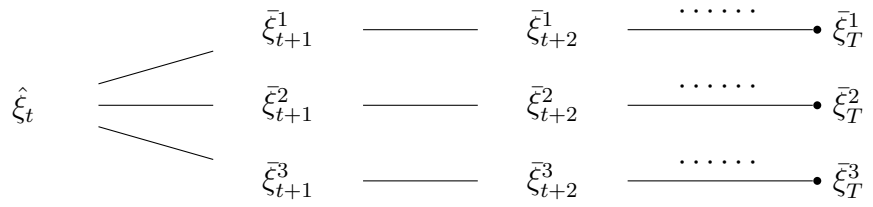
non-Markovian) process $\hat{\xi}_{[T]}$, i.e., $\{u_t, v_t, A_t, B_t, C_t, b_t\}$, for $t = 2, \ldots, T$ (we omit the tilde for ease of notation), we ought to associate a single decision vector $x_1, \ldots, x_T$, generated by solving recursively,

$$\min\ \mathbb{E}_{|\hat{\xi}_{[t]}}\left\{\sum_{i=t}^T \left[\gamma^{i-1}(u_i^\top x_i + v_i^\top y_i)\right]\right\}$$

$$\text{s.t. } A_t x_t + B_t z_t + C_t y_t \geq b_t, z_t = x_{t-1}$$

In order to solve the above problem, we need to discretize it. In section 2.4.2, we approxi-

Figure 2.1: construction of $\tau(\hat{\xi}_t)$



mate the recourse term by iteratively adding cutting planes in backward steps. The rolling horizon based algorithm, which we will refer to it simply as rolling horizon algorithm, discretizes the recourse term directly by dynamically constructing scenario tree $\tau(\hat{\xi}_t)$ rooted at $\hat{\xi}_t$ and its extensive formulation,

$$\begin{aligned} \min \quad & \sum_{n \in \tau(\hat{\xi}_t)} \gamma_n P_n [u_n^\top x_n + v_n^\top y_n] \\ \text{s.t. } & A_n x_n + B_n x_{a(n)} + C_n y_n \geq b_n, \forall n \in \tau(\hat{\xi}_t) \\ & x_{a(\hat{\xi}_t)} = x_{t-1} \end{aligned}$$

where we use $a(\cdot)$ to denote the parent node.

To make the discretized problem tractable, the scenario tree $\tau(\hat{\xi}_t)$ should not be too big. Probably the most intuitive way to do that is to branch the next stage only since the next stage is the immediate future that matters the most. In other words, in stage $t+1$, we generate M samples (nodes) $\bar{\xi}^i_{t+1}, i = 1, \ldots, M$ from the conditional probability distribution of ξ_{t+1} given $\xi_{[t]} = \hat{\xi}_{[t]}$; for stages $s = t+2, \ldots, T$, following each of $\bar{\xi}^i_{s-1}$, we generate only one sample (node) $\bar{\xi}^i_s$ from the conditional probability distribution of ξ_s given $\xi_{[t]} = \hat{\xi}_{[t]}, \xi_{t+1} = \bar{\xi}^i_{t+1}, \ldots, \xi_{s-1} = \bar{\xi}^i_{s-1}$. In this way, the scenario tree grows linearly with the number of stages. Figure 2.1 visualizes an example of such construction of $\tau(\hat{\xi}_t)$ where we chose $M = 3$.

The advantage of the rolling horizon algorithm over the SDDP algorithm is fourfold. First, it only involves linear programming and does not require sophisticated knowledge

of optimization. Second, it dynamically constructs scenario trees so that the true process will not deviate the scenario tree too much. While in the SDDP algorithm, a static scenario tree is constructed beforehand. Third, the time complexity of the rolling horizon algorithm is linear with respect to the number of stages while for the SDDP algorithm this is not guaranteed. And it does not have the problem with the dimension of the state space as SDDP suffers from. Fourth, it does not require any structure of the underlying data process. On the other hand, its disadvantage is also clear. Since it only branches in the next stage, it might be short-sighted. Also, it does not provide a lower bound and thus its performance cannot be guaranteed. Numerical comparisons of these two algorithms are provided in sections 5.1.9 and 5.4.

There are some existing similar rolling horizon based algorithms in the literature, such as the algorithm used in [13]. But as far as we now, the existing algorithms are all implemented on a preconstructed static scenario tree and hence does not solve the true problem.

2.6 Evaluating on the true problem

The aforementioned SDDP solver and extensive solver are used to solve the discretized problem while solving the true problem is our real concern. On the other hand, the rolling horizon solver solves the true problem directly. Ideally, solving the problem would, for example, give us a 2% optimality gap with 95% confidence for the true problem. In practice, quite often it is not possible to get such an answer, especially for large scale problems. The next good answer would be that a solution is reasonably good for the true problem. This is less clear and thus we should be more cautious about it. In this section, we will see how we obtain these two kinds of answers.

2.6.1 Optimality gap for the true problem

As shown in section 2.4.2, by solving a discretized problem, the SDDP solver provides a feasible implementable policy for the true problem. Consequently, we can evaluate the

value of this policy, of the true problem, by Monte Carlo simulations. Similarly as we did in section 2.4.3, by generating samples from the *true* data process instead of the discretized problem, we can construct the approximate $(1-\alpha)$ confidence interval for the expected policy value. The upper end of the CI now provides an upper bound of the optimal value of the true problem with confidence of about $(1-\alpha)$. By contrast, we can not derive an upper bound for the true problem using the extensive solver.

By randomizing SAA discretization (generating different sets of i.i.d samples), we are also able to compute a statistical lower bound for the true problem, for both the SDDP solver and the extensive solver. It is not difficult to show that the expectation of the optimal value of the SAA discretized problem is not larger than the optimal value of the true problem. One can thus get a lower bound for the true problem with a chosen confidence level.

Therefore, it is possible to compute the optimality gap for the true problem by randomizing SAA discretization. However, in practice considered problems often are so large that we cannot afford the cost of solving them several times. Also, note that there is no well-established way to get a lower bound for the true problem when using the Markov chain approximation approach.

On the other hand, the rolling horizon based approach solves the true problem directly and provides us with a feasible implementable policy for the true problem. Therefore, it only gives a statistical upper bound.

We summarize this section by table 2.1.

Table 2.1: Summary of the solvers

problem	solver	implementable	deterministic LB	statistical UB	optimality gap
independent	extensive	✗	✓	✗	✗
	SDDP	✓	✓	✓	✓
	rolling	✓	✗	✓	✗
Markovian	extensive	✗	✗	✗	✗
	SDDP	✓	✗	✓	✗
	rolling	✓	✗	✓	✗

2.6.2 Reasonably good solution to the true problem

The less perfect approach to policy evaluation, discussed below, is to validate the goodness of a discretized problem. This approach is valid for the SDDP solver and the rolling horizon solver.

Consider two implementable feasible policies $\bar{x}_t^{(1)}$ and $\bar{x}_t^{(2)}$, referred to as policy one and policy two respectively. We say policy one is *significantly better* than policy two if the policy value for the true problem of policy one is significantly smaller than policy two in terms of the following one-sided hypothesis testing. Similarly to the way we compared two policies in the last section, we generate M samples from the *true* data process and compute the differences of policy values $D_m(m = 1, \ldots, M)$ (policy 2 subtracting policy 1). Compute its sample mean $\bar{D}_M$ and sample variance S_M^2. Under the null hypothesis of zero mean of the difference, the statistic $T = \frac{\bar{D}_M}{S_M/\sqrt{M}}$ converges in distribution to $\mathcal{N}(0, 1)$. If $T \geq z_\alpha$ we reject the null hypothesis at the confidence level $(1 - \alpha)$ and conclude that policy one is significantly better than policy two. Note that when failing to reject the null hypothesis we do not claim that the true expected value of the considered policies is the same, but only that we cannot detect the difference by this t-test. Note also that this procedure depends on the choice of the number M of generated samples and that its accuracy increases with the increase of the number M.

We say a policy is "reasonably good" if by reasonable efforts, it is not possible to find a significantly better policy. Reasonable efforts may include increasing the granularity of discretization and changing the method of discretization. For example, consider a stage-wise independent continuous true problem. Suppose we have got a policy by solving an SAA problem with 50 samples per stage. One way to verify that the sample size 50 is large enough is to get another policy by solving an SAA problem with 100 sample size. If we compare the two policies by the above test and it fails to reject the null hypothesis with a reasonable number M of generated samples, we argue that our model is reasonably good and it provides a reasonably good solution to the true problem. For another example,

consider a Markovian continuous true problem with a right-hand-side random data process. It is suggested to test both the TS approach and the MCA to discretize the Markovian data process. The two approaches provide us with different discretized problems, possibly giving us significantly different policies (see section 5.1 for an example). These procedures of course are heuristic and should be performed with care.

2.7 Risk averse true problem

So far we only considered optimizing (minimizing) the *expected* policy value. However, for some realizations of the stochastic process the generated costs could be significantly bigger than their average values. This motivates us to introduce a risk averse approach trying to control high quantiles of the random costs. Throughout the thesis, we will focus on coherent law-invariant risk measures.

A risk measure $\varrho : \mathcal{Z} \to \mathbb{R}$, defined on a linear space $\mathcal{Z}$ of random variables, is said to be coherent if it is subadditive (i.e., if $Z, Z' \in \mathcal{Z}$, then $\varrho(Z+Z') \leq \varrho(Z)+\varrho(Z')$), monotone (i.e., if $Z, Z' \in \mathcal{Z}$ are such that $Z \geq Z'$ a.s., then $\varrho(Z) \geq \varrho(Z')$), translation equivariant (i.e., if $Z \in \mathcal{Z}$ and $a \in \mathbb{R}$, then $\varrho(Z + a) = \varrho(Z) + a$), and positively homogeneous (i.e., if $Z \in \mathcal{Z}$ and $t > 0$, then $\varrho(tZ) = t\varrho(Z)$). It is said that ϱ is law invariant if $\varrho(Z) = \varrho(Z')$ for any distributionally equivalent (with respect to the reference probability measure) random variables $Z, Z' \in \mathcal{Z}$.

Axioms of coherent risk measures were introduced in [14], we refer to [15] and [16] for a general discussion of these concepts.

The nested formulation in risk averse setting as follows is the same as in the risk neutral setting except replacing the conditional expectation with conditional analogues $\varrho_{t|\xi_{[t-1]},\eta_{[t-1]}}$

of ϱ_t (recall that the first stage variables are deterministic).

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0}} u_1^\top x_1 + v_1^\top y_1 + \gamma\varrho_{2|\xi_1,\eta_1}\Big[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1}} u_2^\top x_2 + v_2^\top y_2 \\ + \cdots + \gamma\varrho_{T|\xi_{[T-1]},\eta_{[T-1]}}\big[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}}} u_T^\top x_T + v_T^\top y_T\big]\Big].$$

The same applies to the dynamic equations. For example, in the stage-wise independent setting, the dynamic equations with risk measures take the form of,

$$Q_t(x_{t-1},\xi_t) = \min_{\substack{A_tx_t+B_tz_t+C_ty_t\geq b_t\\ z_t=x_{t-1}}} u_t^\top x_t + v_t^\top y_t + \gamma\mathcal{Q}_{t+1}(x_t),$$

where the value functions

$$\mathcal{Q}_{t+1}(x_t) := \begin{cases} \varrho_{t+1|\xi_t}\left[Q_{t+1}(x_t,\xi_{t+1})\right], \text{for } t = T-1,\ldots,1, \\ 0, \text{for } t = T, \end{cases}$$

One popular risk measure is a convex combination of the expectation and Average Value-at-risk (AVaR) risk measure (also called Conditional Value-at-risk, Expected Shortfall, Expected Tail Loss)

$$\varrho_t(\cdot) = (1-\lambda_t)\mathbb{E}(\cdot) + \lambda_t \text{AV@R}_{\alpha_t}(\cdot),\ \alpha_t,\lambda_t\in(0,1),$$

where

$$\text{AV@R}_\alpha(Z) := \inf_{x\in\mathbb{R}}\{x + \alpha^{-1}\mathbb{E}[Z-x]_+\},$$

and its minimum is attained at the so-called Value-at-risk, which is defined as

$$\text{V@R}_\alpha(Z) := \inf\{t : \mathbb{P}(Z\leq t)\geq 1-\alpha\}.$$

In other words, α is the probability that Z (understood as the loss) is larger than $\text{V@R}_\alpha(Z)$.

2.7.1 Solving the risk averse discretized problem

The risk averse problem after discretization can be solved by directly adjusting the gradient computation in line 15 in Algorithm 2. In particular for $\mathbb{E}$-$V@R(\alpha_t, \lambda_t)$ measure, the computation is the following. Sort $\mathscr{V}_{tj}, J = 1, \ldots, N_t$ into ascending order $\mathscr{V}_{t,\pi(1)}, \ldots, \mathscr{V}_{t,\pi(N_t)}$ and let κ be the smallest integer such that $\pi(\kappa) \geq (1-\alpha)N_t$. Then

$$\begin{aligned}
\mathscr{V}_t &= (1-\lambda_t)\frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{V}_{tj} + \lambda_t\left(\mathscr{V}_{t,\kappa} + \frac{1}{\alpha_t N_t}\sum_{j=1}^{N_t}(\mathscr{V}_{tj} - \mathscr{V}_{t,\kappa})_+\right), \\
\mathscr{G}_t &= (1-\lambda_t)\frac{1}{N_t}\sum_{j=1}^{N_t}\mathscr{G}_{tj} + \lambda_t\left(\mathscr{G}_{t,\kappa} + \frac{1}{\alpha_t N_t}\sum_{j=1}^{N_t}(\mathscr{G}_{tj} - \mathscr{G}_{t,\kappa})\mathbb{I}_{\mathscr{V}_{tj}-\mathscr{V}_{t,\kappa}\geq 0}\right),
\end{aligned} \tag{2.12}$$

where $\mathbb{I}(\cdot)$ is the indicator function. In the end, replace line 15 with (2.12).

2.7.2 Policy evaluation

Unfortunately there is no longer easy way to compute the risk-adjusted cost (see a discussion in [17]),

$$u_1^\top x_1 + v_1^\top y_1 + \gamma\varrho_{2|\xi_1,\eta_1}\Big[u_2^\top x_2 + v_2^\top y_2 + \cdots + \gamma\varrho_{T|\xi_{[T-1]},\eta_{[T-1]}}[u_T^\top x_T + v_T^\top y_T]\Big].$$

Therefore, in the forward steps of the SDDP algorithm, we often consider instead the value of its risk neutral component $\sum_{t=1}^{T}\left[\gamma^{t-1}(u_t^\top x_t + v_t^\top y_t)\right]$. As a result, the second stopping criterion, namely evaluation of the optimality gap, is not valid.

In addition, when evaluating the policy generated by risk averse SDDP, the policy value is computed only based on its risk neutral component. In such a way, we can compare policies that generated from solving risk neutral and risk averse problems. One may expect policies generated from risk averse problems have less favourable expected policy value but lower variance of policy values (an example is in section 5.3).

2.8 Multistage stochastic mixed integer programs (MSIP)

Multistage stochastic integer programming (MSIP) asserts integrality restrictions on the feasible region and takes the form of (in the risk neutral setting),

$$\min_{\substack{A_1x_1+B_1z_1+C_1y_1\geq b_1\\ z_1=x_0\\ x_1,y_1\in\mathbb{RZ}}} u_1^\top x_1 + v_1^\top y_1 + \gamma\mathbb{E}_{|\xi_1,\eta_1}\Big[\min_{\substack{A_2x_2+B_2z_2+C_2y_2\geq b_2\\ z_2=x_1\\ x_2,y_2\in\mathbb{RZ}}} u_2^\top x_2 + v_2^\top y_2$$
$$+\cdots+\gamma\mathbb{E}_{|\xi_{[T-1]},\eta_{[T-1]}}\big[\min_{\substack{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T\\ z_T=x_{T-1}\\ x_T,y_T\in\mathbb{RZ}}} u_T^\top x_T + v_T^\top y_T\big]\Big],$$

where $\mathbb{RZ}$ denotes a generic mixed-integer set.

Most of the procedures in the above sections still work in the MSIP setting by simply adding required integrality conditions. For the extensive solver and the rolling horizon solver introduced in section 2.4.1 and section 2.5 respectively, the only difference in the MSIP setting is to construct equivalent deterministic mix-integer programs rather than just linear programs. While for the SDDP solver introduced in section 2.4.2, it is more involved to generalize. This section will demonstrate how to generalize SDDP to stochastic dual dynamic integer programming (SDDiP) in the stage-wise independent setting. Markovian problems can be done similarly.

Again, we first construct a discretized problem by SAA discretization and writing dynamic programming equations in a way similar to equations (2.9). We then use the so-called SDDiP procedure to solve the discretized problem as follows. In the forward steps, the SDDiP solver draws samples from the discretized problem and solves the following mixed integer programs[1] recursively for $t = 1, \ldots, T$,

$$\min_{x_t,y_t\in\mathbb{RZ}} \{u_t^\top x_t + v_t^\top y_t + \gamma\mathfrak{Q}_{t+1}(x_t) : A_tx_t + B_tz_t + C_ty_t \geq b_t, z_t = \bar{x}_{t-1}\},$$

and obtain a feasible solution in each stage $\bar{x}_1, \ldots, \bar{x}_T$. Here $\mathfrak{Q}_{t+1}(\cdot)$ is the current approx-

[1]For simplicity, we omit the superscript i representing the index of iteration.

imation of expected recourse function $\tilde{\mathcal{Q}}_{t+1}(\cdot)$.

In backward steps, for $t = T, \ldots, 2$, for every scenario $j \in N_t$, SDDiP solver solves

$$G^* := \min_{x_t, y_t \in \mathbb{R}\mathbb{Z}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) : A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj}, z_t = \bar{x}_{t-1}\} \quad (2.13)$$

We consider various types of cuts as follows to $\mathfrak{Q}_t(\cdot)$. We say a cut (v_t, k_t) is: (i) *valid*, if $\mathfrak{Q}_t(x) \geq v_t + k_t^\top x$, $\forall x$, (ii) *tight*, if $\mathfrak{Q}_t(\bar{x}_{t-1}) = v_t + k_t^\top \bar{x}_{t-1}$.

2.8.1 Benders' cut

We make the following LP relaxation of (2.13)

$$\min_{x_t, y_t} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) : A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj}, z_t = \bar{x}_{t-1}\}. \quad (2.14)$$

The optimal value $\mathscr{V}_{tj}$ and an optimal dual solution $\mathscr{G}_{tj}$ corresponding to constraints $z_t = \bar{x}_{t-1}$ serve as coefficients of a Benders' cut of $\tilde{\mathcal{Q}}_{t+1}(\cdot)$ at $\bar{x}_{t-1}$ for scenario j. The Benders' cut is then given by,

$$\mathfrak{Q}_t(x) \geq (\frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{G}_{tj}^\top)(x - \bar{x}_{t-1}) + \frac{1}{N_t}\sum_{j=1}^{N_t} \mathscr{V}_{tj}.$$

The Benders' cut is valid but not necessarily tight.

2.8.2 Strengthened Benders' cut

Consider the following Lagrangian dual problem of (2.13),

$$\begin{aligned} & g^* := \max_{\pi}\{g(\pi)\}, \\ & g(\pi) := \min_{x_t, y_t \in \mathbb{R}\mathbb{Z}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) - \pi^\top (z_t - \bar{x}_{t-1}) \\ & : A_{tj}x_t + B_{tj}z_t + C_{tj}y_t \geq b_{tj}\}. \end{aligned} \quad (2.15)$$

Note that the outer problem is unconstrained with respect to π, hence every π will provide a valid cut. In particular, we let $\pi = \mathscr{G}_{tj}$ and solve the following program

$$\min_{x_t, y_t \in \mathbb{R}\mathbb{Z}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) - \mathscr{G}_{tj}^\top z_t : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}\}.$$

Suppose its optimal value is $\mathscr{V}_{tj}^{SB}$. The strengthened Benders' cut is then given by

$$\mathfrak{Q}_t(x) \geq (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj})^\top x + \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}^{SB}.$$

Note that the strengthened Benders' cut is paralleled and superior to its corresponding Benders' cut. The strengthened Benders' cut is valid but still not necessarily tight.

2.8.3 Lagrangian cut

The Lagrangian cut is obtained by solving the dual problem (2.15). Since $g(\pi)$ is a concave piece-wise linear function, we can use the level method [18] to solve it, as shown in Algorithm 5.

It is recommended to set the step size as 0.2929 [18]. We don't know the optimal value of (2.15) in general. Thus in Algorithm 5, we keep updating our estimate of g^* by the set of cutting planes. In the next section, we will see binarization of the state space makes $G^* = g^*$ and thus

$$\mathfrak{Q}_t(\bar{x}_{t-1}) = \left(\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG} \right)^\top \bar{x}_{t-1} + \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}^{LG},$$

assuring that Lagrangian cuts are tight. The stopping criteria of the level method are similar to those in the SDDP solver.

2.8.4 Binarization

In order to binarize an MSIP, we make the following additional assumption.

Algorithm 5 Level method to compute Lagrangian cuts at stage t

Given a fixed step size λ and state $\bar{x}_{t-1}$.
for $j = 1, \ldots, N_t$ **do**
  Initialize $\pi_1 = \mathscr{G}_{tj}$, $k = 1$.
  Initial approximation $\Psi^0 = \{\tau : \tau(\cdot) \leq u\}$
  while no stopping criterion is met **do**
    Solve the inner problem: $g(\pi_k) := \min_{x_t, y_t \in \mathbb{RZ}} \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \mathfrak{Q}_{t+1}(x_t) - \pi_k^\top (z_t - \bar{x}_{t-1}) : A_{tj} x_t + B_{tj} z_t + C_{tj} y_t \geq b_{tj}\}$ and obtain an optimal solution $x_t = x_t^{(\pi_k)}, z_t = z_t^{(\pi_k)}$.
    Update cut collection $\Psi^k \leftarrow \{\tau \in \Psi^{k-1} | \tau(\pi_k) \leq g(\pi_k) - (\pi - \pi_k)^T (z_t^{(\pi_k)} - \bar{x}_{t-1})\}$
    Solve: $g^* = \max_\pi \{\tau(\pi) \in \Psi^k\}$
    Update best so far solution π_k^* such that $g_* := g(\pi_k^*) = \max_{i=1,\ldots,k} \{g(\pi_i)\}$
    Update level $l_k = \lambda g_* + (1 - \lambda) g^*$
    Update $\pi_{k+1} = \text{argmin}_\pi \big\{ \big\| \pi - \pi_k^* \big\|_2^2 \big| g(\pi_i) - (\pi - \pi_i)^T (z_t^{(\pi_i)} - \bar{x}_{t-1}) \geq l_k, \forall i = 1, 2, \ldots, k \big\}$
    $k = k + 1$
  end while
  $\mathscr{G}_{tj}^{LG} = \pi^*, \mathscr{V}_{tj}^{LG} = g(\pi^*) - (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG})^\top \bar{x}_{t-1}$
end for
$\mathfrak{Q}_t \leftarrow \{\theta \in \mathfrak{Q}_t, \theta(x) \geq (\frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}^{LG})^\top x + \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}^{LG}\}$

Assumption 2 The polyhedral set $\{x_t = (x_{t,1}, \ldots, x_{t,n_t}) \in \mathbb{RZ}^{n_t} : A_t x_t + B_t x_{t-1} + C_t y_t \geq b_t\}$ is bounded over time, where n_t is the dimension of the state space and $\mathbb{RZ}^{n_t}$ is a mix-integer set of dimension n_t. That is, there exist $U_t = (u_{t,1}, \ldots, u_{t,n_t})$ and $V_t = (v_{t,1}, \ldots, v_{t,n_t})$ such that $U_t \leq x_t \leq V_t$ in the element-wise sense, i.e., $u_{t,i} \leq x_{t,i} \leq v_{t,i}$, $i = 1, \ldots, n_t$, for all x_t in the polyhedral set, $t = 1, \ldots, T$.

If Assumption 2 holds, we approximate the decimal variable $x_{t,i}$ by a binary variable vector $\mathbf{x}_{t,i}^{(2)}$ as follows:

$$
\begin{aligned}
& 10^{k_{t,i}} (x_{t,i} - u_{t,i}) = H_{t,i}^\top x_{t,i}^{(2)}, \\
& H_{t,i} = (2^0, \ldots, 2^{B_{t,i}-1})^\top, \\
& B_{t,i} = \big\lfloor \log_2 [10^{k_{t,i}} (v_{t,i} - u_{t,i})] \big\rfloor + 1, \\
& x_{t,i}^{(2)} \in \{0, 1\}^{B_{t,i}},
\end{aligned}
$$

where $k_{t,i}$ is the number of decimal places of accuracy for each dimension i in stage t and is set to zero if $x_{t,i} \in \mathbb{Z}$. Thus the state variable variable $x_t = (x_{t,1}, \ldots, x_{t,n_t})$ is approximated by binary state variable $x_t^{(2)} = (x_{t,1}^{(2)\top}, \ldots, x_{t,n_t}^{(2)\top})$ of length $B_t = \sum_{i=1}^{n_t} B_{t,i}$, so as the local copy variable z_t. We then have the *binarized discretized problem*,

$$\begin{aligned} \tilde{Q}_t(x_{t-1}^{(2)}, \xi_t) = \min \; & u_t^\top x_t + v_t^\top y_t + \gamma \tilde{\mathcal{Q}}_{t+1}^{(2)}(x_t^{(2)}) \\ \text{s.t. } & A_t x_t + B_t z_t + C_t y_t \geq b_t, \\ & z_t^{(2)} = x_{t-1}^{(2)}, \\ & 10^{k_{t,i}}(x_{t,i} - u_{t,i}) = H_{t,i}^\top x_{t,i}^{(2)}, \\ & 10^{k_{t,i}}(z_{t,i} - u_{t,i}) = H_{t,i}^\top z_{t,i}^{(2)}, \\ & x_{t,i}^{(2)} \in \{0,1\}^{B_{t,i}}, \end{aligned} \tag{2.16}$$

for $t = T, \ldots, 1$. Here $x_0^{(2)}$ is the binary representation of initial value x_0 and the binarized approximate expected cost-to-go is defined as

$$\tilde{\mathcal{Q}}_{t+1}^{(2)}(x_t^{(2)}) := \begin{cases} \dfrac{1}{J_{t+1}} \displaystyle\sum_{j=1}^{J_{t+1}} \tilde{Q}_{t+1}^{(2)}(x_t^{(2)}, \xi_{t+1}^{(2)j}), \text{ for } t = T-1, \ldots, 1, \\ 0, \text{ for } t = T. \end{cases}$$

Similar to (2.15), we can make Lagrangian dual problem of (2.16). As shown in [19], there is no gap between (2.16) and its Lagrangian dual problem; the SDDiP algorithm with Lagrangian cuts guarantees to produce an optimal solution to the binarized discretized problem in a finite number of iterations with probability one.

It was given in [19] the upper bound of $k_i, \forall i$, to obtain an ϵ optimal solution the discretized problem by solving the binarized discretized problem. In practice such bound is rarely applicable and our real primal interest is solving the true problem. We now illustrate how to evaluate the obtained policy on the true problem. The SDDiP solver produces approximation of $\tilde{\mathcal{Q}}_2^{(2)}(\cdot), \ldots, \tilde{\mathcal{Q}}_T^{(2)}(\cdot)$, denoted by $\mathfrak{Q}_2^{(2)}(\cdot), \ldots, \mathfrak{Q}_T^{(2)}(\cdot)$, and a feasible first

stage solution $\bar{x}_1^{(2)}$. This generates a feasible implementable policy $\bar{x}_t(\xi_{[t]})$ for the true problem as follows. For any realization of the process $\{u_t, v_t, A_t, B_t, C_t, b_t\}, t = 2, \ldots, T$, we associate a decision vector $x_1 (= H\bar{x}_1^{(2)}), x_2, \ldots, x_T$ generated by solving recursively,

$$
\begin{aligned}
\min \quad & u_t^\top x_t + v_t^\top y_t + \gamma \mathfrak{Q}_{t+1}^{(2)}(x_t^{(2)}) \\
\text{s.t.} \quad & A_t x_t + B_t z_t + C_t y_t \geq b_t, \\
& z_t = x_{t-1}, \\
& 0 \leq 10^{k_{t,i}}(x_{t,i} - u_{t,i}) - H_{t,i}^\top x_{t,i}^{(2)} < 1, \\
& x_{t,i}^{(2)} \in \{0,1\}^{B_{t,i}},\ i = 1, \ldots, n,\ t = 2, \ldots, T.
\end{aligned}
$$

Again, since Assumption 1 holds, it provides a feasible implementable policy for the true problem.

CHAPTER 3
MSPPY, A PYTHON PACKAGE FOR MULTISTAGE STOCHASTIC PROGRAMMING

3.1 Introduction

Since the publication of the pioneering paper by [7] on the Stochastic Dual Dynamic Programming (SDDP) method, considerable efforts have been made to apply/enhance the algorithm in both academia and industry. Especially in recent years, open-source software packages, including SDDP.jl [20], StructDualDynProg.jl [21], StochDynamicProgramming.jl [22], FA-ST [23], StOpt [24], based on the SDDP algorithm, have sprung up and are able to solve a wide variety of problems. A comprehensive comparison of these packages provided in [20] shows that SDDP.jl has the least restrictions on the underlying data process. It also could be mentioned that the SDDiP.jl [25], which is an extension of SDDP.jl, implements the integer version of SDDP proposed in [19].

All of the above mentioned packages are built for the case when the underlying data process is finite discrete. Their typical user input is a finite list of scenarios or a Markov chain, and all of the implementations and analysis are based on that assumption. However, quite often it is natural to model the data process as having a continuous distribution with an infinite number of possible realizations. (We refer to that as the 'true' problem). This leads to two implications. First, in order to use those packages, users themselves need to discretize the true process. This procedure is quite involved if the modeled process is Markovian. Second, these packages do not provide a solution and a way of analyzing the 'true' problem. This is somewhat disappointing since a poorly discretized model may have a considerable deviation from the naturally modeled problem. When solving multistage stochastic integer programs (MSIP), binarization is sometimes made to make cuts tight,

which further obscures the situation by bringing another layer of approximation. Users of these packages should be extremely cautious about what they are actually solving.

This chapter presents the so-called MSPPy package we developed in [1], which makes a step forward in removing these concerns[1].

The current version of the MSPPy package uses the Gurobi solver and the Python interface. It requires **Python 3+** and **gurobipy 7+** [26]. The entire package is based on object oriented programming, making codes readable and further development easy. It has five main modules, *msp*, *sp*, *discretize*, *solver* and *evaluation*.

The *msp*, *sp*, *discretize* modules are used to build and discretize the true problem. The *msp* module contains an MSLP base class, corresponding to a multistage stochastic program. The *sp* module contains a StochasticModel base class representing an individual stage problem. The *discretize* module includes the MCA approach to discretize Markovian data processes.

The *solver* module is used to construct solver objects to solve a discretized problem. An extensive solver object can be used to solve an MSLP/MSIP discretized problem. An SDDP(SDDiP) solver object can be used to solve an MSLP(MSIP) discretized problem. A rolling solver object can be used to solve directly an MSLP/MSIP true problem.

The *evaluation* module includes two classes, *Evaluation* and *EvaluationTrue*, to understand policies obtained by SDDP/SDDiP solver. An *Evaluation* object evaluates a policy on the discretized problem while an *EvaluationTrue* object evaluates a policy on the true problem.

3.2 Using the SDDP solver to solve MSLP

In this section, we illustrate a standardized procedure to solve MSLP using the SDDP solver. We will see how to use the SDDiP solver to solve MSIP in section 3.3.

[1]The package is subjected to changes. For the latest update and more detail, please go to the MSPPy package website https://github.com/lingquant/msppy and the documentation website https://msppy.readthedocs.io/en/latest

3.2.1 Step one: build the true problem

To build the true problem, the first step is to create an MSLP object. An MSLP object is composed of a list of StochasticModel objects. Users are then required to fill in each StochasticModel object. The StochasticModel class is a stochastic version of the gurobipy.Model that built in the Gurobi library. It allows users to directly write into a stochastic model rather than treating the deterministic counterpart and uncertainties separately. In order to achieve it while staying close to the gurobipy syntax, the MSPPy package encapsulates the gurobipy.Model and its randomness. Hence, all things that work on gurobipy.Model will work on Stochastic Model. In addition, four routines from gurobipy are overwritten and several new routines are created for modeling convenience. The four overwritten routines as shown in the snippet below, *addVar*, *addVars*, *addConstr*, *addConstrs*, include additional arguments called *uncertainty* and *uncertainty_dependent* to incorporate stage-wise independent data process and Markov process. Uncertainties that appear in the objective, u_i, v_i, can be added along with adding related variables (by *addVar* or *addVars*). Uncertainties that appear in the constraints, A_i, B_i, C_i, b_i, can be added along with adding related constraints (by *addConstr* or *addConstrs*).

```
m = StochasticModel()
m.addVars(...,uncertainty, uncertainty_dependent)
m.addVar(..., uncertainty, uncertainty_dependent)
m.addConstrs(...,uncertainty, uncertainty_dependent)
m.addConstr(...., uncertainty, uncertainty_dependent)
```

Two new routines are added in order to include state variable(s). Local copy variable(s) will be added correspondingly behind the scenes. In the following snippet, now is a reference to the added state variable(s) and past is a reference to the corresponding local copy variable(s).

```
now, past = m.addStateVars(...,uncertainty,uncertainty_dependent)
```

```
now, past = m.addStateVar(...,uncertainty,uncertainty_dependent)
```

Using the above routine to add multiple stage-wise independent uncertainties sequentially inherently assume the added uncertainties are independent. The MSPPy package provides another function *add_continuous_uncertainty* that is able to incorporate an uncertainty that follows a multivariate distribution,

```
m.add_continuous_uncertainty(uncertainty, locations)
```

Stage-wise independent problem

We first indicate procedures and examples to build stage-wise independent MSLP. Stage-wise independent uncertainties should be specified by scenarios or random variable generators.

Step 1.1 (Stage-wise independent discrete/continuous) Create an MSLP instance by specifying number of time periods T. This will create a list of T empty StochasticModel objects.

Step 1.2 (Stage-wise independent discrete/continuous) Run a for loop and fill in the T StochasticModel objects, namely,

$$\min_{A_1x_1+B_1z_1+C_1y_1\geq b_1} u_1^\top x_1 + v_1^\top y_1, \cdots, \min_{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T} u_T^\top x_T + v_T^\top y_T$$

Note that the balance constraints $z_t = x_{t-1}$ will be added behind the scenes.

The following snippet specifies a three stage MSLP with stage-wise independent finite discrete uncertainty. In every stage, a state variable x_t and a local copy z_t are added. The random objective coefficients u_2, u_3 are independent and take 1.5 or 0.5 with equal probability.

```
uncertainty = [[1],[1.5,0.5],[1.5,0.5]]
discrete = MSLP(T = 3)
for t in range(3):
```

```
    m = discrete[t]
    now, past = m.addStateVar(uncertainty=uncertainty[t])
```

The following snippet specifies a three stage MSLP with a stage-wise independent continuous uncertainty. The random objective coefficients u_2, u_3 are independent and follow the log-normal distribution $\log \mathcal{N}(0, 1)$.

```
def f(random_state):
    return random_state.lognormal(0,1)
continuous = MSLP(T = 3)
for t in range(3):
    m = continuous[t]
    if t == 0:
        now, past = m.addStateVar()
    else:
        now, past = m.addStateVar(uncertainty=f)
```

Markovian MSLP

We now demonstrate procedures and examples to build Markovian MSLP. A nonhomogeneous Markov chain process should be specified by Markov state spaces and transition matrices for each stage. A Markovian continuous process should be specified by a sample path generator. Each dimension of the Markovian uncertainty is then added to the right place (either in the objective or in constraints) to the constructed MSLP.

Step 1.1(Markovian discrete/continuous) Create an MSLP instance by specifying number of time periods T and a sample path generator/Markov chain. This will create a list of T empty StochasticModel objects.

Step 1.2(Markovian discrete/continuous) Run a for loop and fill in the T Stochastic-

Model objects, namely,

$$\min_{A_1x_1+B_1z_1+C_1y_1\geq b_1} u_1^\top x_1 + v_1^\top y_1, \cdots, \min_{A_Tx_T+B_Tz_T+C_Ty_T\geq b_T} u_T^\top x_T + v_T^\top y_T$$

Again, the balance constraints $z_t = x_{t-1}$ will be added behind the scenes.

The following snippet formulates a three stage MSLP with Markov chain objective coefficient u_t. The initial Markov state $u_1 = 1$. The Markov states and transition matrices for stage two and stage three are given by,

$$\begin{array}{c} \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 3 \end{array} \\ \begin{pmatrix} 0.2 & 0.8 \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \begin{pmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \end{pmatrix} \end{array}$$

```
mc = MSLP(T = 3)
mc.add_MC_uncertainty(
    Markov_states=[[1],[2,3],[4,5]],
    transition_matrix=[[[1]],[[0.2,0.8]],[[0.3,0.7],[0.4,0.6]]]
)
for t in range(3):
    m = MC[t]
    now, past = m.addStateVar(uncertainty_dependent=0)
```

Note that in this case the Markovian uncertainty is unidimensional, the dimension index *uncertainty_dependent* is simply 0 (it starts with zero in Python). In other words, the first dimension of the Markovian process appears at the objective coefficient of the variable *now*.

The following snippet formulates a three stage MSLP with Markovian objective coefficient u_t where u_t follows an autoregressive time series model: $u_t = 0.5u_{t-1} + \epsilon_t$, $\epsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0,1)$ with initial value $u_0 = 0$.

```
markovian = MSLP(T = 3)
def sample_path_generator(random_state, size):
    u = numpy.zeros([size,T,1])
    for t in range(1,T):
        u[:,t,:] = 0.5 * u[:,t-1,:]
            + random_state.normal(0,1,size=size))
    return a
markovian.add_Markovian_uncertainty(sample_path_generator)
for t in range(3):
    m = Markovian[t]
    now, past = m.addStateVar(uncertainty_dependent=0)
```

Finally, users can specify a risk measure. Also, for a stage-wise independent finite discrete uncertainty, users can specify the probability mass function that it follows.

3.2.2 Step two: discretize the true problem

Stage-wise independent uncertainties are discretized by SAA (see section 2.2.1). The following snippet makes a discretization with 100 i.i.d samples for each stage except stage one.

```
continuous.discretize(n_samples=100)
```

Markovian uncertainties can be discretized by MCA (see section 2.2.3). The following snippet uses the SA method and 1000 sample paths to train an approximating Markov chain with 10 Markov states from stage two on.

```
markovian.discrete(n_Markov_states=10, n_sample_paths=1000,
    method='SA')
```

3.2.3 Step three: solve the discretized problem

We can now use the SDDP solver to solve the discretized problem with certain stopping criterion. First, create an SDDP instance solver,

```
example_SDDP = SDDP(example)
```

Next, call the *solve* method to run the SDDP algorithm with one of the three types of stopping criterion introduced in section 2.4.3.

The first type of stopping criteria

The first type of stopping criteria is an upper limit of certain characteristics among time, the number of iterations and the number of iterations that the deterministic bound does not change.

```
example_SDDP.solve(max_iterations, max_time,
    max_stable_iterations)
```

The second type of stopping criteria

We evaluate the policy every *freq_evaluations* iterations by running *n_simulations* Monte Carlo simulations. If the gap becomes not larger than *tol*, the algorithm will be stopped.

```
example_SDDP.solve(freq_evaluations, n_simulations, tol)
```

The third type of stopping criteria

We compare the policy every *freq_comparisons* iterations by computing the CI of the difference of the expected policy values. If the upper end of CI becomes not larger than *tol_diff*, the algorithm will be stopped.

```
example_SDDP.solve(freq_comparisions, n_simulations, tol_diff)
```

SDDP solver utilizes the **multiprocessing** library to make parallelization. Parallelization can be switched on by passing to the solve method the number of forward and backward steps to run per iteration and the number of processes to parallelize the jobs.

```
example_SDDP.solve(n_steps, n_processes)
```

The obtained policy by SDDP solver is a first stage state solution and approximations of value functions (cutting planes). Writing all models to disk reveal all the cutting planes in a readable way.

```
example.first_stage_solution
example.write()
```

3.2.4 Step four: evaluate the SDDP policy on the true problem

The MSPPy package evaluates the SDDP policy by Monte Carlo simulation. As an example, the following snippet queries the confidence interval of the simulated policy values of the true problem and queries the solution of a variable called ”purchase”.

```
example_result = EvaluationTrue(example)
example_result.run(n_simulations, query=["purchase"])
example_result.CI
example_result.solution["purchase"]
```

3.3 Using the SDDiP solver to solve MSIP

The five essential steps to solve MSIP by the SDDiP solver are:

Step one: build the true problem.

Step two: discretize the true problem.

Step three: binarize the discretized problem. (This step can be skipped.)

Step four: solve the (binarized) discretized problem.

Step five: evaluate the computed policy.

Compared to solving MSLP, for step two and step five, the usage syntax is exactly the same as shown in section 3.2; for step one, users only need to change the key word from MSLP to MSIP. Hence, in this section we will only go through steps three and four.

3.3.1 Step three: binarize the discretized problem

The following snippet binarizes the discretized problem for stages before *bin_stage* (exclusive). Precision parameter k dictates the number of decimal places of accuracy.

```
MSIP().binarize(bin_stage, precision)
```

3.3.2 Step four: solve the (binarized) discretized problem

The three types of stopping criterion mentioned in section (3.2.3) are still valid here. If Lagrangian cuts are used, stopping criterion and specifics of the level method can be specified.

```
SDDiP().solve(cuts=['LG'], level_step_size, level_tol)
```

The SDDiP solver should be called with arguments specifying the types of cuts. The following three cut patterns are supported.

For a cyclical cut pattern, cuts are added cyclically. The following snippet dictates for every ten iterations, adding Benders' cut for the first three, adding Strengthened Benders' cut for the next three, and adding Lagrangian cut for the last four.

```
SDDiP().solve(cuts=['B','SB','LG'], pattern={'cycle':[3,3,4]})
```

For a barrier-in cut pattern, cuts are added from certain iteration. The following snippet dictates adding Benders' cut from beginning, adding Strengthened Benders' cut from iteration 10, and adding Lagrangian cut from iteration 20.

```
SDDiP().solve(cuts=['B','SB','LG'], pattern={'in':[0,10,20]})
```

For some problems with exceptionally long horizon, sometimes the integrality conditions in the far future may contribute negligibly to the objective. The following routine does an LP relaxation to the stage models after stage 100.

```
SDDiP().solve(cuts=['B','SB','LG'], relax_stage=100)
```

3.4 Using the extensive solver to solve MSLP/MSIP

The three essential steps to solve MSLP/MSIP by the extensive solver are:

Step one: build the true problem.

Step two: discretize the true problem.

Step three: solve the discretized problem.

Recall that the extensive solver does not provide an implementable policy for the true problem. Therefore, there is no step four to evaluate on the true problem. Compared to the SDDP/SDDiP solver, for step one and step two, the usage syntax is exactly the same as shown in the previous sections. For step three, the extensive solver just constructs an equivalent deterministic LP/IP and solves it.

```
Extensive().solve()
```

3.5 Using the rolling solver to solve MSLP/MSIP

The two essential steps to solve MSLP/MSIP by the rolling solver are:

Step one: build the true problem.

Step two: solve the true problem and evaluate the obtained policy.

Recall that the rolling solver is able to solve the true problem directly. As a result, there is no need to discretize the problem and we can combine the procedure of solving and evaluation. In step two, for Markovian MSLP/MSIP, apart from the sample path generator, the number of branches and the conditional probability distribution used to construct dynamic scenario trees should be specified. Evaluation parameters should also be inputted. For ex-

ample, for the true problem with underlying process $u_t = 0.5u_{t-1} + \epsilon_t$, where $\epsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0, 1)$ constructed in section 3.2.1, the following snippet solves it and evaluates the obtained policy,

```
def conditional_dist(random_state, prev, t):
    noise = random_state.normal(0,1)
    return 0.5 * prev + noise
Rolling().solve(n_branches, n_simulations,
    conditional_dist=conditional_dist)
```

CHAPTER 4

PERIODICAL MULTISTAGE STOCHASTIC PROGRAMS

4.1 Introduction

We have seen in the previous chapters that the development of approximate approaches to dynamic programming, the SDDP method in particular, somehow mitigates the curse of dimensionality we introduced in chapter 1. Empirically, it has more or less linear complexity with respect to the number of stages. But the complexity with respect to the dimension of the state space is often exponential. In the case of two stages, the SDDP algorithm is equivalent to the Kelley's algorithm [27]. A worst-case analysis of the Kelley's algorithm showed in [28] is that the following convex problem:

$$\min_{x\in\mathbb{R}^{n+1}} f(x) \text{ s.t. } \|x\| \leq 1,$$

with $f(x) := \max\left\{\|x\|^2 - x_{n+1}^2, |x_{n+1}|\right\}$ requires $O\left(\frac{ln\epsilon^{-1}}{2ln2}(\frac{2}{\sqrt{3}})^{n-1}\right)$ iterations to obtain an ϵ- optimal solution for an n-dimensional problem. In other words, the required number of iterations grows exponentially with the increase of the dimensionality. Coupling such difficulty with multiple stages, the multistage case is even worse.

In this chapter, we present an approach to improve the SDDP that we proposed in [29]. In some applications the considered multistage stochastic programs have a periodical behavior. *The main result in this chapter is to demonstrate that in such cases it is possible to drastically reduce the number of stages by introducing a periodical analog of the so-called Bellman equations, used in MDP and SOC, and consequently applying a variant of the SDDP algorithm.* In the next section we give a general formulation of infinite horizon risk averse stochastic programming problems. The main theoretical developments are presented in section 4.3 where we introduce Bellman equations adjusted to the periodical behavior of

the considered infinite horizon problems. In section 4.4 we briefly discuss statistical inference of the Sample Average Approximation (SAA) approach to discretization of possibly continuous distributions of the data process. An SDDP type algorithm for solving the obtained periodical dynamical equations is described in section 4.5. Numerical experiments and remarks with the proposed variant of SDDP algorithm applied to the Brazilian interconnected power system (cf., [30]) will be provided in section 5.1.7. Section 4.7 gives a brief discussion of an extension to a Markovian setting.

4.2 Multistage programming

Consider the following risk averse multistage stochastic programming problem, given in the nested form,

$$\begin{array}{rcl} \min\limits_{\substack{A_1x_1=b_1\\ x_1\in\mathcal{X}_1}} f_1(x_1) & + & \gamma\rho_{|\xi_1}\Big(\min\limits_{\substack{B_2x_1+A_2x_2=b_2\\ x_2\in\mathcal{X}_2}} f_2(x_2,\xi_2)+\cdots \\ & & +\gamma\rho_{|\xi_{[T-1]}}\big(\min\limits_{\substack{B_Tx_{T-1}+A_Tx_T=b_T\\ x_T\in\mathcal{X}_T}} f_T(x_T,\xi_T)\big)\Big), \end{array} \tag{4.1}$$

Here $\gamma \in (0,1)$ is the discount factor, $\xi_t \in \mathbb{R}^d$, $t = 1,...,T$, is a random data process, $f_t : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ are cost functions and can be non-linear, $b_t = b_t(\xi_t)$ are right side vectors, $B_t = B_t(\xi_t)$ and $A_t = A_t(\xi_t)$ are matrices of appropriate dimensions and $\mathcal{X}_t \subset \mathbb{R}^n$ are closed sets. Here $B_1 = 0$ and hence the term B_1x_0 at the first stage is omitted. The number of stages T considered can be finite or it can be that $T = \infty$. In order to make formulation (4.1) precise, certain regularity conditions should be introduced; we will discuss this later.

Assuming that the risk measure ϱ is law invariant and coherent, for finite time horizon T, problem (4.1) can be written as

$$\min_{\pi\in\Pi} f_1(x_1^\pi) + \mathcal{R}_T\left(\textstyle\sum_{t=2}^T \gamma^{t-1} f_t(x_t^\pi,\xi_t)\right), \tag{4.2}$$

Where $f_1(x_1, \xi_1) = f_1(x_1)$, and $\mathcal{R}_T$ is the respective nested risk measure

$$\mathcal{R}_T(\cdot) := \rho_{|\xi_1}\big(\cdots \rho_{|\xi_{[T-1]}}(\cdot)\big).$$

That is, for a given policy $\pi \in \Pi$, $Z_t := f_t(x_t(\xi_{[t]}), \xi_t)$ and $S_T := \sum_{t=2}^{T} \gamma^{t-1} Z_t$ we have that

$$\mathcal{R}_T(S_T) = \gamma\rho_{|\xi_1}\big(Z_2 + \cdots + \gamma\rho_{|\xi_{[T-1]}}(Z_T)\big).$$

We are going to deal with limit risk measures as $T \to \infty$. In order to proceed we need to impose certain boundedness conditions. Suppose that random variables Z_t are bounded, say $|Z_t| \leq \kappa$ a.s. for all t. Then

$$\begin{aligned} \mathcal{R}_{T+1}(S_{T+1}) &= \gamma\rho_{|\xi_1}\big(Z_2 + \cdots + \gamma\rho_{|\xi_{[T-1]}}(Z_T + \gamma\rho_{|\xi_{[T]}}(Z_{T+1})\big) \\ &\leq \gamma\rho_{|\xi_1}\big(Z_2 + \cdots + \gamma\rho_{|\xi_{[T-1]}}(Z_T + \gamma\kappa)\big) \leq \mathcal{R}_T(S_T) + \gamma^T\kappa. \end{aligned}$$

Also

$$\begin{aligned} \mathcal{R}_T(S_T) &= \gamma\rho_{|\xi_1}\big(Z_2 + \cdots + \gamma\rho_{|\xi_{[T-1]}}(Z_T + \gamma\rho_{|\xi_{[T]}}(Z_{T+1}) - \gamma\rho_{|\xi_{[T]}}(Z_{T+1})\big) \\ &\geq \gamma\rho_{|\xi_1}\big(Z_2 + \cdots + \gamma\rho_{|\xi_{[T-1]}}(Z_T + \gamma\rho_{|\xi_{[T]}}(Z_{T+1})\big) - \gamma^T\kappa, \end{aligned}$$

and hence $|\mathcal{R}_{T+1}(S_{T+1}) - \mathcal{R}_T(S_T)| \leq \gamma^T\kappa$. And so on for $T < T'$ we have that

$$\big|\mathcal{R}_{T'}(S_{T'}) - \mathcal{R}_T(S_T)\big| \leq \kappa\gamma^T \sum_{k=0}^{\infty} \gamma^k = \frac{\kappa\gamma^T}{1-\gamma}, \tag{4.3}$$

and hence $\mathcal{R}_2(S_2), ...,$ is a Cauchy sequence. Letting $T \to \infty$ we have then that the following limit does exist

$$\mathcal{R}_\infty(S_\infty) := \lim_{T\to\infty} \rho_{|\xi_1}\big(\cdots \rho_{|\xi_{[T-1]}}(S_T)\big). \tag{4.4}$$

In particular if $\varrho := \mathbb{E}$ is given by the expectation operator, with $\rho_{|\xi_{[t]}} = \mathbb{E}_{|\xi_{[t]}}$ being the corresponding conditional expectations, then $\mathcal{R}_T$ is also the respective expectation operator. In that case we refer to (4.2) as the risk neutral formulation of the multistage program.

- Unless stated otherwise we assume that the process ξ_t is stagewise independent, i.e., for $t = 1, ...,$ random vector ξ_{t+1} is independent of $\xi_{[t]}$.

When the number T of stages is finite, the nested formulation (4.1) leads to the following dynamic programming equations (cf., [16]). At stage $t = T$ the value (cost-to-go) function $V_T(x_{T-1}, \xi_T)$ is given by the optimal value of the problem

$$\begin{array}{ll} \min\limits_{x_T \in \mathcal{X}_T} & f_T(x_T, \xi_T) \\ \text{s.t.} & B_T x_{T-1} + A_T x_T = b_T. \end{array} \tag{4.5}$$

Note that variable x_T in (4.5) is viewed now as a deterministic vector $x_T \in \mathbb{R}^{n_T}$ and should satisfy the corresponding feasibility constraints. At stages $t = T - 1, ..., 2$, the value function $V_t(x_{t-1}, \xi_t)$ is given by the optimal value of the problem

$$\begin{array}{ll} \min\limits_{x_t \in \mathcal{X}_t} & f_t(x_t, \xi_t) + \gamma \mathcal{V}_{t+1}(x_t) \\ \text{s.t.} & B_t x_{t-1} + A_t x_t = b_t, \end{array} \tag{4.6}$$

with

$$\mathcal{V}_{t+1}(x_t) = \varrho\big(V_{t+1}(x_t, \xi_{t+1})\big). \tag{4.7}$$

At the first stage the following problem should be solved

$$\begin{array}{ll} \min\limits_{x_1 \in \mathcal{X}_1} & f_1(x_1) + \gamma \mathcal{V}_2(x_1) \\ \text{s.t.} & A_1 x_1 = b_1. \end{array} \tag{4.8}$$

4.3 Infinite horizon periodical setting

Consider the infinite horizon setting (i.e., $T = \infty$) where problem (4.2) has the following periodical behavior with period $\mathsf{m} \in \mathbb{N}$. That is, let us make the following assumptions leading to stationary solutions.

(A1) The random vectors ξ_t and $\xi_{t+\mathsf{m}}$ have the same distribution, with support $\Xi \subset \mathbb{R}^d$, for $t \geq 2$ (recall that ξ_1 is deterministic).

(A2) The functions $b_t(\cdot)$, $B_t(\cdot)$, $A_t(\cdot)$ and $f_t(\cdot, \cdot)$ have period m, i.e., are the same for $t = \tau$ and $t = \tau + \mathsf{m}$, $t = 2, ...$, and the sets $\mathcal{X}_t$ are nonempty and $\mathcal{X}_t = \mathcal{X}_{t+\mathsf{m}}$ for all t.

(A3) The functional $\varrho : \mathcal{Z} \to \mathbb{R}$ is a law invariant coherent risk measure, defined on space $\mathcal{Z} = L_p(\Xi, \mathcal{B}, P)$ for some $p \in [1, \infty]$.

(A4) The functions $f_t(\cdot, \cdot)$ are bounded, i.e., there is $\kappa > 0$ such that $|f_t(x_t, \xi_t)| \leq \kappa$ for all $(x_t, \xi_t) \in \mathcal{X}_t \times \Xi$ and all t.

(A5) For every $x_{t-1} \in \mathcal{X}_{t-1}$ the set $\{x_t \in \mathcal{X}_t : B_t(\xi_t)x_{t-1} + A_t(\xi_t)x_t = b_t(\xi_t)\}$ is nonempty for a.e. $\xi_t \in \Xi$ and $t \geq 2$.

In assumption (A3), $\mathcal{B}$ is the Borel sigma algebra of subsets of Ξ and P is a probability measure on $(\Xi, \mathcal{B})$, referred to as the reference measure. Unless stated otherwise all probabilistic statements are made with respect to the reference measure P. In particular, we sometimes write $\mathbb{E}_P$ to emphasize that the expectation is taken with respect to the distribution P. Assumption (A5) usually is referred to as the relatively complete recourse.

Under these assumptions the value functions $\mathcal{V}_t(\cdot)$ and $\mathcal{V}_{t+\mathsf{m}}(\cdot)$ of dynamic equations (4.6) - (4.7) are the same for all $t \geq 2$. This leads to the following periodical variant of Wald-Bellman (WB) equations (often simply referred to as Bellman equations in Optimal Control and MDP) for the value functions $\mathcal{V}_2(\cdot), ..., \mathcal{V}_{\mathsf{m}+1}(\cdot)$,

$$\mathcal{V}_\tau(x_{\tau-1}) = \varrho(V_\tau(x_{\tau-1}, \xi_\tau)), \tag{4.9}$$

with

$$V_\tau(x_{\tau-1},\xi_\tau) = \inf_{x_\tau\in\mathcal{X}_\tau}\left\{f_\tau(x_\tau,\xi_\tau)+\gamma\mathcal{V}_{\tau+1}(x_\tau) : B_\tau x_{\tau-1}+A_\tau x_\tau = b_\tau\right\}, \tag{4.10}$$

for $\tau = 2,...,\mathsf{m}+1$, and $\mathcal{V}_{\mathsf{m}+2}$ replaced by $\mathcal{V}_2$ for $\tau = \mathsf{m}+1$ (note that ξ_τ in (4.9) is viewed as a *random* vector, while in (4.10) it is treated as a vector valued variable). Consequently for $t \geq \mathsf{m}+2$ the corresponding value functions are defined recursively as $V_t(\cdot,\xi_t) = V_{t-\mathsf{m}}(\cdot,\xi_t)$, and hence $\mathcal{V}_t(\cdot) = \mathcal{V}_{t-\mathsf{m}}(\cdot)$. At the first stage problem (4.8) should be solved.

In order to show that equations (4.9) - (4.10) have a solution we proceed as follows (cf., [31]). Let $\mathbb{B}(\mathcal{X}_t)$ be the space of bounded functions $g : \mathcal{X}_t \to \mathbb{R}$. Note that $\mathbb{B}(\mathcal{X}_t)$, equipped with the sup-norm $\|g\| = \sup_{x\in\mathcal{X}_t}|g(x)|$, is a Banach space. Consider the space $\mathfrak{B} := \mathbb{B}(\mathcal{X}_1)\times\cdots\times\mathbb{B}(\mathcal{X}_\mathsf{m})$ equipped with the max-norm $\|\mathfrak{g}\|_\mathfrak{B} := \max\{\|g_2\|,...,\|g_{\mathsf{m}+1}\|\}$, for $\mathfrak{g} = (g_2,...,g_{\mathsf{m}+1}) \in \mathfrak{B}$ composed from functions $g_t : \mathcal{X}_{t-1} \to \mathbb{R}$. Moreover consider mapping $\mathfrak{T} : \mathfrak{B} \to \mathfrak{B}$ defined for $\mathsf{m} \geq 2$ as

$$\mathfrak{T}(\mathfrak{g})(\mathfrak{x}) := \left(\varrho(\Psi_{g_3}(x_1,\xi_2)),...,\varrho(\Psi_{g_{\mathsf{m}+1}}(x_{\mathsf{m}-1},\xi_\mathsf{m})),\varrho(\Psi_{g_2}(x_\mathsf{m},\xi_{\mathsf{m}+1}))\right), \tag{4.11}$$

where $\mathfrak{x} = (x_1,...,x_\mathsf{m}) \in \mathfrak{X}$, with $\mathfrak{X} := \mathcal{X}_1\times\cdots\times\mathcal{X}_\mathsf{m}$, and

$$\Psi_{g_{\tau+1}}(x_{\tau-1},\xi_\tau) := \inf_{x_\tau\in\mathcal{X}_\tau}\left\{f_\tau(x_\tau,\xi_\tau)+\gamma g_{\tau+1}(x_\tau) : B_\tau x_{\tau-1}+A_\tau x_\tau = b_\tau\right\}, \tag{4.12}$$

for $\tau = 2,...,\mathsf{m}$, and $g_{\mathsf{m}+2}(\cdot) = g_2(\cdot)$ for $\tau = \mathsf{m}+1$. The case of $\mathsf{m} = 1$ is discussed in Remark 4.3.1 below.

Assumptions A(4) and A(5) imply that for any $g_{\tau+1} \in \mathbb{B}(\mathcal{X}_\tau)$ the functions $\Psi_{g_{\tau+1}}(x_{\tau-1},\xi_\tau)$ are bounded. In order to verify that $\varrho\left(\Psi_{g_{\tau+1}}(x_{\tau-1},\xi_\tau)\right)$ are well defined we still need to verify that $\Psi_{g_{\tau+1}}(x_{\tau-1},\cdot)$ are measurable. We implicitly assume this. When the set Ξ is finite this holds automatically. In general we refer to [31],[32] for a discussion of such

measurability issues. Then the mapping $\mathfrak{T}$ is well defined, i.e., for any $\mathfrak{g} \in \mathfrak{B}$ its image $\mathfrak{T}(\mathfrak{g})$ belongs to the space $\mathfrak{B}$. We have that $\mathfrak{V} = (\mathcal{V}_2, ..., \mathcal{V}_{\mathsf{m}+1})$ is a solution of equations (4.9)–(4.10) iff it is a fixed point of the mapping $\mathfrak{T}$. Let us show that $\mathfrak{T}$ is a contraction mapping with respect to the norm $\|\cdot\|_{\mathfrak{B}}$ (in that we follow [31, Section 4.2]). Note that the space $\mathfrak{B}$, with the norm $\|\cdot\|_{\mathfrak{B}}$, is a Banach space.

Proposition 1 *For every $\mathfrak{g}, \mathfrak{g}' \in \mathfrak{B}$ the following inequality holds*

$$\|\mathfrak{T}(\mathfrak{g}) - \mathfrak{T}(\mathfrak{g}')\|_{\mathfrak{B}} \leq \gamma \|\mathfrak{g} - \mathfrak{g}'\|_{\mathfrak{B}}. \tag{4.13}$$

Proof. It follows from monotonicity of ϱ that $\mathfrak{T}$ has the following monotonicity property: if $\mathfrak{g}, \mathfrak{g}' \in \mathfrak{B}$ are such that $\mathfrak{g} \succeq \mathfrak{g}'$, then $\mathfrak{T}(\mathfrak{g}) \succeq \mathfrak{T}(\mathfrak{g}')$ (by writing $\mathfrak{g} \succeq \mathfrak{g}'$ we mean that $g_t(\cdot) \geq g'_t(\cdot)$ for $t = 2, ..., \mathsf{m}+1$). It follows from translation equivariance of ϱ that $\mathfrak{T}$ has the following property of constant shift: for any $\mathfrak{g} \in \mathfrak{B}$ and $c \in \mathbb{R}$ it follows that $\mathfrak{T}(\mathfrak{g} + c\mathfrak{e}) = \mathfrak{T}(\mathfrak{g}) + \gamma c\mathfrak{e}$, where $\mathfrak{e} \in \mathfrak{B}$ is such that $\mathfrak{e}(\mathfrak{x}) = (1, ..., 1)$ for any $\mathfrak{x} \in \mathfrak{X}$. These two properties imply that $\mathfrak{T}$ is a contraction mapping. Indeed, consider $\mathfrak{g}, \mathfrak{g}' \in \mathfrak{B}$ and let $\delta := \|\mathfrak{g} - \mathfrak{g}'\|_{\mathfrak{B}}$. Then we have that $\mathfrak{g} + \delta\mathfrak{e} \succeq \mathfrak{g}'$, and hence by monotonicity of $\mathfrak{T}$ it follows that $\mathfrak{T}(\mathfrak{g} + \delta\mathfrak{e}) \succeq \mathfrak{T}(\mathfrak{g}')$. Applying the constant shift property we obtain that $\mathfrak{T}(\mathfrak{g}) + \gamma\delta\mathfrak{e} \succeq \mathfrak{T}(\mathfrak{g}')$. That is $\gamma\delta\mathfrak{e} \succeq \mathfrak{T}(\mathfrak{g}') - \mathfrak{T}(\mathfrak{g})$. The opposite inequality can be shown in the similar way, and hence (4.13) follows. ∎

The contraction property of $\mathfrak{T}$ implies the following.

Theorem 1 *Under assumptions* (A1)–(A5) *the following holds:* (i) *There exists unique set of functions $\mathcal{V}_2, ..., \mathcal{V}_{\mathsf{m}+1}$ satisfying the WB equations* (4.9)-(4.10). (ii) *For any $\mathfrak{g}_0 \in \mathfrak{B}$, the sequence of functions defined iteratively $\mathfrak{g}_{k+1} = \mathfrak{T}(\mathfrak{g}_k)$, $k = 0, 1, ...$, converges (in the norm $\|\cdot\|_{\mathfrak{B}}$) to $\mathfrak{V} = (\mathcal{V}_2, ..., \mathcal{V}_{\mathsf{m}+1})$ as $k \to \infty$.* (iii) *If, moreover, for all t the sets $\mathcal{X}_t$ are convex and $f_t(x_t, \xi_t)$ are convex in $x_t \in \mathcal{X}_t$, then the functions $\mathcal{V}_t : \mathcal{X}_{t-1} \to \mathbb{R}$, $t \geq 2$, are convex.*

Proof. Since the mapping $\mathfrak{T}$ is a contraction mapping, the assertions (i) and (ii) follow by the Banach Fixed Point Theorem. Suppose now that $\mathcal{X}_t$ and $f_t(\cdot, \xi_t)$ are convex, and consider $\mathfrak{g} \in \mathfrak{B}$ with convex component functions $g_{\tau+1}$. It follows then that the functions $\Psi_{g_{\tau+1}}(\cdot, \xi_\tau)$ are convex. Consequently by convexity and monotonicity of ϱ, we have that $\varrho\left(\Psi_{g_{\tau+1}}(\cdot, \xi_\tau)\right)$ are convex. Consequently the components of $\mathfrak{T}^k(\mathfrak{g})(\cdot)$ are convex for all k and hence the fixed point of $\mathfrak{T}$ has convex components. This proves (iii). ∎

Let $\mathcal{V}_2, ..., \mathcal{V}_{\mathsf{m}+1}$ be the value functions satisfying the WB equations (4.9)-(4.10), and for $t \geq \mathsf{m}+2$ the corresponding value functions be defined recursively as $\mathcal{V}_t(\cdot) = \mathcal{V}_{t-\mathsf{m}}(\cdot)$. The minimizers

$$\begin{aligned} &\bar{x}_1 \in \arg\min_{x_1 \in \mathcal{X}_1} \big\{ f(x_1) + \gamma \mathcal{V}_2(x_1) : A_1 x_1 = b_1 \big\}, \\ &\bar{x}_t \in \arg\min_{x_t \in \mathcal{X}_t} \big\{ f_t(x_t, \xi_t) + \gamma \mathcal{V}_{t+1}(x_t) : B_t(\xi_t)\bar{x}_{t-1} + A_t(\xi_t) x_t = b_t(\xi_t) \big\},\ t \geq 2, \end{aligned} \tag{4.14}$$

give an optimal policy of the corresponding infinite horizon problem. Note that for $t = 2, ...,$ the minimizer $\bar{x}_t$ is a function of $\bar{x}_{t-1}$ and ξ_t. The respective optimal value is given by the optimal value of the first stage problem.

Remark 4.3.1 For $\mathsf{m} = 1$ the data is the same for all t, therefore in that case we remove the subscript t from the data functions. The WB equations for the value function $\mathcal{V} = \mathcal{V}_2$ become

$$\begin{aligned} \mathcal{V}(x) &= \varrho(V(x, \xi)), \\ V(x, \xi) &= \inf_{x' \in \mathcal{X}} \big\{ f(x', \xi) + \gamma \mathcal{V}(x') : B(\xi)x + A(\xi)x' = b(\xi) \big\}, \end{aligned} \tag{4.15}$$

where ξ denotes random vector having distribution of ξ_t. Solution of these equations are given by the fixed point of the mapping $\mathfrak{T} : \mathbb{B}(\mathcal{X}) \to \mathbb{B}(\mathcal{X})$ defined as

$$\mathfrak{T}(g)(x) := \varrho\left(\Psi_g(x, \xi)\right) \tag{4.16}$$

with

$$\Psi_g(x,\xi) := \inf_{x' \in \mathcal{X}} \big\{ f(x',\xi) + \gamma g(x') : B(\xi)x + A(\xi)x' = b(\xi) \big\}. \tag{4.17}$$

Under the specified assumptions the mapping $\mathfrak{T}$ is a contraction mapping and the assertions of Theorem 1 do apply.

4.4 Statistical inference

For the sake of simplicity let us consider first the case of the period length $\mathsf{m} = 1$, with the corresponding WB equations given in (4.15). In order to evaluate the value function $\mathcal{V}(\cdot)$ numerically we need to discretize the (possibly continuous) distribution of the random vector ξ. We discuss this in the framework of the (infinite horizon) problem (4.2) with $T = \infty$.

Let $\hat{P}$ be an approximation of the reference distribution P and $\hat{\varrho}$ be the corresponding risk measure. In the risk neutral case we have that $\varrho = \mathbb{E}_P$ and then $\hat{\varrho} = \mathbb{E}_{\hat{P}}$. Consider mapping $\mathfrak{T}$, defined in (4.16), corresponding to the original (true) problem and its approximation counterpart $\hat{\mathfrak{T}} : \mathbb{B}(\mathcal{X}) \to \mathbb{B}(\mathcal{X})$ defined in the similar way

$$\hat{\mathfrak{T}}(g)(x) := \hat{\varrho}\left(\Psi_g(x,\xi)\right). \tag{4.18}$$

Under the specified regularity conditions, both mappings $\mathfrak{T}$ and $\hat{\mathfrak{T}}$ are contraction mappings. Let $\mathcal{V}$ and $\hat{\mathcal{V}}$ be their fixed points, i.e., $\mathfrak{T}(\mathcal{V}) = \mathcal{V}$ and $\hat{\mathfrak{T}}(\hat{\mathcal{V}}) = \hat{\mathcal{V}}$. Then by using the contraction property of $\hat{\mathfrak{T}}$ we have

$$\begin{aligned} \|\mathcal{V} - \hat{\mathcal{V}}\| &= \|\mathfrak{T}(\mathcal{V}) - \hat{\mathfrak{T}}(\hat{\mathcal{V}})\| = \|\mathfrak{T}(\mathcal{V}) - \hat{\mathfrak{T}}(\mathcal{V}) + \hat{\mathfrak{T}}(\mathcal{V}) - \hat{\mathfrak{T}}(\hat{\mathcal{V}})\| \\ &\leq \|\mathfrak{T}(\mathcal{V}) - \hat{\mathfrak{T}}(\mathcal{V})\| + \gamma\|\mathcal{V} - \hat{\mathcal{V}}\|. \end{aligned}$$

It follows that

$$\|\mathcal{V}-\hat{\mathcal{V}}\| \leq (1-\gamma)^{-1}\|\mathfrak{T}(\mathcal{V})-\hat{\mathfrak{T}}(\mathcal{V})\| = (1-\gamma)^{-1}\left\|\varrho\left(\Psi_{\mathcal{V}}(x,\xi)\right)-\hat{\varrho}\left(\Psi_{\mathcal{V}}(x,\xi)\right)\right\|. \quad (4.19)$$

Consider now the Sample Average Approximation (SAA) approach to discretization of the true problem. That is, a random sample $\xi^1, ..., \xi^N$, of iid realizations of random vector ξ, is generated by Monte Carlo sampling techniques and distribution P is approximated by the corresponding empirical distribution $\hat{P}_N = N^{-1}\sum_{j=1}^N \delta_{\xi^j}$. We can view the associated risk measure $\hat{\varrho}_N$ and value function $\hat{\mathcal{V}}_N$ as the SAA estimates of ϱ and $\mathcal{V}$, respectively. In particular, in the risk neutral case, when $\varrho = \mathbb{E}_P$, we have that

$$\hat{\varrho}_N\left(\Psi_{\mathcal{V}}(x,\xi)\right) = \mathbb{E}_{\hat{P}_N}\left[\Psi_{\mathcal{V}}(x,\xi)\right] = \frac{1}{N}\sum_{j=1}^N \Psi_{\mathcal{V}}(x,\xi^j). \quad (4.20)$$

Assuming that the set $\mathcal{X}$ is compact, $\Psi_{\mathcal{V}}(x,\xi)$ is continuous in $x \in \mathcal{X}$ and is dominated by an integrable function, we have by the uniform Law of Large Numbers (LLN) (e.g., [16, Theorem 7.53]) that

$$\lim_{N\to\infty}\left\|\mathbb{E}_P\left[\Psi_{\mathcal{V}}(x,\xi)\right]-\mathbb{E}_{\hat{P}_N}\left[\Psi_{\mathcal{V}}(x,\xi)\right]\right\| = 0 \text{ w.p.1.} \quad (4.21)$$

There is also a uniform LLN for general law invariant coherent risk measures ϱ (cf., [33]). It follows then by (4.19) that $\hat{\mathcal{V}}_N(\cdot)$ converges w.p.1 to $\mathcal{V}(\cdot)$ uniformly on $\mathcal{X}$. This in turn implies that the optimal value and first stage optimal solutions of the SAA problem converge w.p.1 to their counterparts of the (true) problem (e.g., [16, Theorem 5.3]).

In the risk neutral setting, when $\varrho = \mathbb{E}_P$, there are also available results for uniform in $x \in \mathcal{X}$ rates of convergence of $\mathbb{E}_{\hat{P}_N}\left[\Psi_{\mathcal{V}}(x,\xi)\right]$ to $\mathbb{E}_P\left[\Psi_{\mathcal{V}}(x,\xi)\right]$ (e.g., [16, Section 7.2.10]). By (4.19) this implies that for $\varepsilon > 0$, under appropriate regularity conditions, probability of the event $\{\|\mathcal{V}-\hat{\mathcal{V}}_N\| \geq \varepsilon\}$ converges to zero exponentially fast with increase of the sample size N. It also follows from those results that in order to solve the true problem

with accuracy $\varepsilon > 0$ by solving the SAA problem with accuracy $\delta \in [0, \varepsilon)$, for a given confidence one needs a sample size N of order $O\left((\varepsilon - \delta)^{-2}\right)$ (cf., [16, Section 5.3.2]).

In the general setting of $\mathsf{m} \geq 1$ we can proceed in the similar way. The true (marginal) distribution of every ξ_t is approximated by generating an iid sample $\xi_t^1, ..., \xi_t^N$, say of the same size N for every $t = 2, ..., \mathsf{m}+1$. (Recall that the process ξ_t is assumed to be stagewise independent.) Consequently the value functions $\mathcal{V}_t(x_{t-1})$ are approximated by their sample average approximations

$$\hat{\mathcal{V}}_{t,N}(x_{t-1}) = \hat{\varrho}_N\left(\Psi_{\mathcal{V}_{t+1}}(x_{t-1}, \xi_t)\right), \ t = 2, ..., \mathsf{m}+1,$$

by replacing the true (original) distribution of vectors ξ_t in WB equations (4.9) - (4.10) with their respective empirical distributions based on the generated samples. Similar to (4.19) by using the max-norm $\|\cdot\|_{\mathfrak{B}}$, we have that for $t = 2, ..., \mathsf{m}+1$,

$$\|\mathcal{V}_t - \hat{\mathcal{V}}_{t,N}\| \leq (1-\gamma)^{-1} \max_{2 \leq \tau \leq \mathsf{m}+1} \left\|\varrho\left(\Psi_{\mathcal{V}_{\tau+1}}(x_{\tau-1}, \xi_\tau)\right) - \hat{\varrho}_N\left(\Psi_{\mathcal{V}_{\tau+1}}(x_{\tau-1}, \xi_\tau)\right)\right\|, \quad (4.22)$$

with $\mathcal{V}_{\mathsf{m}+2} = \mathcal{V}_2$.

Under mild regularity conditions (in particular assuming compactness of the sets $\mathcal{X}_t$) a uniform LLN can be applied to the empirical risk measures $\hat{\varrho}_N$ (cf., [33],[16, Section 7.2.6]). It follows then that the optimal value and first stage optimal solutions of the SAA problem converge w.p.1 to their counterparts of the (true) problem. In the risk neutral case, when $\varrho = \mathbb{E}$, it is also possible to apply a general analysis of sample complexity of the SAA approach in the multistage setting (we refer to [16, Section 5.8.2] for the derivations and discussion of required regularity conditions). Again for a given accuracy $\varepsilon > 0$ and $\delta \in [0, \varepsilon)$ one needs the sample size N (per stage) of order $O\left((\varepsilon - \delta)^{-2}\right)$. Note however that the total number of scenarios (sample paths) of the constructed SAA problem is $\mathcal{N} = N^{\mathsf{m}}$. Therefore the corresponding number of *scenarios* $\mathcal{N}$ is of order $O\left((\varepsilon - \delta)^{-2\mathsf{m}}\right)$.

4.5 Cutting plane algorithm

There are different approaches to computing value function, satisfying the WB equations, which were suggested in the literature (see, e.g., [31]). For a recent discussion of limitations of classical approaches we can refer to [34]. In this section we discuss a cutting planes approach to approximation of the value functions. This is somewhat similar to the Stochastic Dual Dynamic Programming (SDDP) method (introduced in [35]), which became popular for solving multistage linear (convex) stochastic programs. Such type of algorithms for solving the WB equations in Optimal Control, with $\mathsf{m} = 1$, were recently considered in [36] and [34].

Let us consider first the case of $\mathsf{m} = 1$ (see Remark 4.3.1); the general periodic case will be discussed in sections below. We assume that the distribution of the random vector ξ is discretized, say by the SAA method, so that the set $\Xi = \{\xi^1, ..., \xi^N\}$ is finite, equipped with probabilities $p_j > 0$, $j = 1, ..., N$. We assume that $f(x, \xi)$ is convex in x, in particular it can be linear $f(x, \xi) = c(\xi)^\top x$, and the set $\mathcal{X}$ is convex, and hence functions $V(\cdot, \xi)$ and $\mathcal{V}(\cdot)$ are convex. For now we consider the risk neutral case, when $\varrho = \mathbb{E}$; later we will comment how this can be extended to general coherent risk measures. The WB equations can be written here as

$$\begin{aligned} \mathcal{V}(x) &= \sum_{j=1}^{N} p_j V^j(x), \\ V^j(x) &= \min_{x' \in \mathcal{X}} \left\{ f^j(x') + \gamma \mathcal{V}(x') : B^j x + A^j x' = b^j \right\}, \ j = 1, ..., N, \end{aligned} \tag{4.23}$$

where $f^j(x) = f(x, \xi^j)$, $B^j = B(\xi^j)$, $A^j = A(\xi^j)$, $b^j = b(\xi^j)$, and p_j is the probability of scenario j. Given a current piecewise linear (under) approximation $\underline{\mathcal{V}}(\cdot)$ of function $\mathcal{V}(\cdot)$ and a trial point $x \in \mathcal{X}$, the next cutting plane is constructed by computing a subgradient g^j, at x, of the current estimate of value function

$$\underline{V}^j(x) = \min_{x' \in \mathcal{X}} \left\{ f^j(x') + \gamma \underline{\mathcal{V}}(x') : B^j x + A^j x' = b^j \right\} \tag{4.24}$$

for $j = 1, ..., N$. This subgradient is obtained by solving the respective dual problem. If $f(x, \xi^j) = (c^j)^\top x$ is linear and $\mathcal{X}$ is a polyhedral set given by a finite number of affine constraints, then problem (4.24) is a linear programming problem. Then the respective subgradient of $\mathcal{V}(\cdot)$ at the point x is estimated as $\sum_{j=1}^N p_j g^j$. There are different strategies for generating trial point(s) at each iteration, we will discuss this in section 4.6.1 below.

For an estimate $\underline{\mathcal{V}}(\cdot)$ of the value function, consider policy

$$\bar{x}_t \in \arg\min_{x_t \in \mathcal{X}} \left\{ f(x_t, \xi_t) + \gamma \underline{\mathcal{V}}(x_t) : B(\xi_t)\bar{x}_{t-1} + A(\xi_t)x_t = b(\xi_t) \right\},\ t = 1, ..., \tag{4.25}$$

with $\bar{x}_0 = 0$ and $f(x_1, \xi_1) = f_1(x_1)$. This is a feasible policy for the respective infinite horizon problem. For a sample path (realization) $\xi_1, ...,$ of the random process the point estimate of corresponding value of this policy is $\sum_{t=1}^\infty \gamma^{t-1} f_t(\bar{x}_t, \xi_t)$. Hence the value of this policy can be estimated by averaging these point estimates for M randomly generated sample paths (this can be applied to the discretized or the original distribution of the random process). This will give an upper estimate of the optimal value of problem. This is similar to the forward step of the SDDP algorithm.

Note that for finite sum $\sum_{t=1}^T \gamma^{t-1} f_t(\bar{x}_t, \xi_t)$ we can estimate the respective error as (compare with (4.3))

$$\left| \sum_{t=1}^\infty \gamma^{t-1} f_t(\bar{x}_t, \xi_t) - \sum_{t=1}^T \gamma^{t-1} f_t(\bar{x}_t, \xi_t) \right| = \left| \sum_{t=T+1}^\infty \gamma^{t-1} f_t(\bar{x}_t, \xi_t) \right| \leq \frac{\kappa \gamma^T}{1-\gamma}, \tag{4.26}$$

where κ is such that $|f_t(\bar{x}_t, \xi_t)| \leq \kappa$.

Similar procedure can be applied to a law invariant coherent risk measure ϱ. The only difference is how a subgradient of $\underline{V}^j(x)$ is computed (cf., [30]). Unfortunately for general coherent risk measures it is not clear how to compute an upper bound for the optimal value of the considered risk averse problem by the randomization method, discussed above for the risk neutral problem. Construction of the upper bound by an outer approximation was suggested in [37]. That procedure is based on a certain discretization of the set $\mathcal{X}$;

as a consequence the number of discretization points needed to achieve a given accuracy typically grows exponentially with increase of the dimension of $x \in \mathcal{X}$.

In the next section we discuss an implementation of SDDP type algorithm, referred to as the periodical SDDP algorithm, applied to the periodical WB equations, for general period $\mathsf{m} \geq 1$ and risk measures.

4.6 Periodical SDDP (PSDDP)

In this section we discuss the proposed periodical variant SDDP algorithm applied to periodical infinite horizon problems. We follow the formalism, and apply software, described in chapter 2 and chapter 3. That is, consider risk functional $\mathcal{R}_\infty$, defined in (4.4), and the following problem

$$\min_{\pi \in \Pi} \mathcal{R}_\infty \left(\sum_{t=1}^{\infty} \gamma^{t-1} (u_t^\top x_t + v_t^\top y_t) \right), \tag{4.27}$$

with decision variables (x_t, y_t) and respective policies $\pi \in \Pi$ satisfying the constraints

$$B_t x_{t-1} + A_t x_t + C_t y_t \geq b_t. \tag{4.28}$$

In that approach x_t can be viewed as state variables and y_t as local (control) variables similar to the model used in the Optimal Control. The difference from the Optimal Control approach is that the state variables x_t are not determined completely by the local variables y_t and the minimization in (4.27) is performed jointly in x_t and y_t. Anyway in the above formulation the corresponding value functions are functions of state variables and do not depend on y_t.

Suppose that the above problem has a periodical behavior with period $\mathsf{m} \geq 1$. The stagewise independent stochastic processes $u_t, v_t, A_t, B_t, C_t, b_t$ are discretized into finite set Ω_t for each stage $t = 2, \ldots, \mathsf{m}+1$ (recall that the first stage data u_1 and v_1 are deterministic). The discretized problem is solved by a periodical version of the SDDP algorithm (see Algorithm 6 below).

Algorithm 6 Periodical SDDP algorithm for stagewise independent infinite horizon problem with a discount factor $\gamma \in (0,1)$ and a period $\mathsf{m} \geq 1$

Given sample size N_t and discretization $\Omega_t = \{u_t^j, v_t^j, A_t^j, B_t^j, C_t^j, b_t^j\}_{1 \leq j \leq N_t}$, for $t = 2, \ldots, \mathsf{m}+1$
Initial approximation of value functions: $\hat{\mathcal{V}}_2^{(0)}, \ldots, \hat{\mathcal{V}}_{\mathsf{m}+1}^{(0)}, \hat{\mathcal{V}}_{\mathsf{m}+2}^{(0)} (= \hat{\mathcal{V}}_2^{(0)})$
Initialize: $i = 1$, LB = -∞
while no stopping criterion is met **do**
(Forward Step)
  for $t = 1, \ldots, T (T \geq \mathsf{m}+1)$ **do**
    if $t = 1$ **then** $t' = 1$
    else
      $t' \equiv t \pmod{\mathsf{m}}$, where $t' \in \{2, \ldots, \mathsf{m}+1\}$
      Draw a sample $(u_{t'}, v_{t'}, A_{t'}, B_{t'}, C_{t'}, b_{t'})$ from $\Omega_{t'}$
    end if
    $\tilde{x}_t, \tilde{y}_t = \text{argmin} \{u_{t'}^\top x_t + v_{t'}^\top y_t + \gamma \hat{\mathcal{V}}_{t'+1}^{(i-1)}(x_t) : A_{t'} x_t + B_{t'} z_t + C_{t'} y_t \geq b_{t'}, z_t = \tilde{x}_{t-1}\}$
  end for
  Trial point selection: $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{\mathsf{m}})$ selected from candidates $(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_T)$
(Backward Step)
  for $t = \mathsf{m}+1, \ldots, 2$ **do**
    for $j = 1, \ldots, N_t$ **do**
      if $t = \mathsf{m}+1$ **then** $i' = i-1$
      else $i' = i$
      end if
      Solve $\min \{u_{tj}^\top x_t + v_{tj}^\top y_t + \gamma \hat{\mathcal{V}}_{t+1}^{(i')}(x_t) : A_t^j x_t + B_t^j z_t + C_t^j y_t \geq b_t^j, z_t = \bar{x}_{t-1}\}$ and get optimal value $\mathscr{V}_{tj}$ and dual solution $\mathscr{G}_{tj}$ corresponding to constraint $z_t = \bar{x}_{t-1}$
    end for
    Compute coefficients of a new cutting plane, $\mathscr{V}_t$ and $\mathscr{G}_t$, from $\mathscr{V}_{tj}$ and $\mathscr{G}_{tj}$. In particular, for risk neutral measure: $\mathscr{V}_t := \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{V}_{tj}, \mathscr{G}_t := \frac{1}{N_t} \sum_{j=1}^{N_t} \mathscr{G}_{tj}$
    $\hat{\mathcal{V}}_t^{(i)} \leftarrow \{\theta \in \hat{\mathcal{V}}_t^{(i-1)}, \theta \geq \mathscr{G}_t (x_{t-1} - \bar{x}_{t-1}) + \mathscr{V}_t\}$
  end for
  LB $= \min\{u_1^\top x_1 + v_1^\top y_1 + \gamma \hat{\mathcal{V}}_2^{(i)}(x_1) : A_1 x_1 + B_1 z_1 + C_1 y_1 \geq b_1, z_1 = x_0\}$
  $i = i+1$
end while

4.6.1 Construction of trial points

There is a delicate issue of choosing trial points where the value functions are updated by construction of the respective cutting planes. This is related to the question of eventual convergence of the algorithm. Recall that the aim here is solving the corresponding infinite horizon problem (with the discount factor $\gamma < 1$). The WB equations describe the value functions for this infinite horizon problem. Available proofs of convergence of standard SDDP algorithms, for finite horizon problems, are quite technical (see [38],[39], and references therein). Careful derivation of such proofs for the considered framework is left for a future research.

When the number of scenarios (sample paths) is finite, such proofs of convergence are based on the property that the considered algorithm eventually goes through *every* scenario. Of course when the number of scenarios is astronomically large, this can not happen in a reasonable time. Therefore although such proofs have a theoretical interest, they do not guarantee convergence to an arbitrary small precision for larger problems. Therefore although such theoretical developments give general guidelines, in practice various heuristics are used. We used the following heuristics trying to mimic the standard SDDP approach, this worked reasonably well.

In the forward step of SDDP algorithm, by iteratively solving the first T stage problems, we obtain solutions of state variables, denoted by $\tilde{x}_t$, for each stage $t = 1, \ldots, T$. Those solutions are considered as candidates of trial points for stage 2 to stage $\mathsf{m} + 1$ in the backward step. Various trial points selection methods can be considered. The method we adopt here is as follows. Consider all consecutive m-stage periods for the T stages, i.e., $\big\{(s, s+1, \ldots, s+\mathsf{m}-1) : \ s = 1, \mathsf{m}+1, 2\mathsf{m}+1 \ldots, s+\mathsf{m}-1 \leq T\big\}$. Randomly pick one of the cycles $(\hat{s}, \hat{s}+1, \ldots, \hat{s}+\mathsf{m}-1)$. Trial points are then selected as $\bar{x}_k = \tilde{x}_{s_0+k-1}, k = 1, \ldots, \mathsf{m}$.

The backward step then solves each stage problem at those trial points $\bar{x}_k, k = \mathsf{m}, \ldots, 1$, from stage $\mathsf{m} + 1$ to stage 2 backwardly and iteratively adds a corresponding cutting plane

to the previous stage. The cutting plane obtained from stage 2 is also added to stage $\mathsf{m}+1$ since $\hat{\mathcal{V}}_{\mathsf{m}+2}(\cdot) = \hat{\mathcal{V}}_2(\cdot)$. In the end, by solving the first stage problem, the backward step is able to produce a lower bound for the infinite horizon problem. We add local copy variables $z_t, t = 1, \ldots, T$, to simplify dual computation and modeling process.

4.6.2 Policy evaluation

After solving the discretized problem, the obtained policy is feasible and implementable for infinite number of stages. The classical SDDP on the other hand, produces a policy that is implementable up to a finite number of stages. Nevertheless, we can still compare the two approaches by evaluating the obtain policies up to some finite number of stages both for the discretized problem and the true problem (by "true" we mean the original problem with possibly continuous distributions of the random variables). The evaluation of the periodical SDDP can be done by running the forward step in Algorithm 6 multiple times.

In the risk neutral setting, the finite summation of individual discounted stage costs $\sum_{t=1}^{T} \gamma^{t-1} \left[u_{t'}^\top \bar{x}_t + v_{t'}^\top \bar{y}_t\right]$, for sufficiently large T, can be regarded as an approximation of the policy value for $T = \infty$ (the error of such approximation is given by (4.26)). Therefore, we are able to construct confidence interval of the (approximated) policy value. If we replace the discretized set Ω_t by the true distribution of the stochastic processes, we are also able to construct confidence interval of the policy value for the true problem. In the end, the optimality gap of the discretization problem is given by the percentage difference between the upper end of the confidence interval and the lower bound. It is also possible to compute the optimality gap of the true problem by randomizing the discretized problem. Stopping criteria and parallelization technique for the classical SDDP are also valid here. All of the evaluation techniques are quite similar to the classical SDDP. For more details, we direct readers to chapter 2.

4.7 Markovian setting

Let us finally mention the following extension to a Markovian setting. For the sake of simplicity consider the case of $\mathsf{m} = 1$ (see Remark 4.3.1). Suppose that the process η_t is Markovian and each η_t has the same marginal distribution. For example η_t can be stationary first order autoregressive process $\eta_t = \mu + \Phi\eta_{t-1} + \varepsilon_t$, or (after discretization) η_t can be a stationary homogeneous Markov chain. Let η and η' be random vectors with η having distribution of η_t, and η' having conditional distribution of η_{t+1} given η_t. For the Markovian process, the WB equations take the form (compare with equations (4.15))

$$\begin{aligned} \mathcal{V}(x,\eta) &= \varrho_{|\eta}[V(x,\eta')], \\ V(x,\eta) &= \min_{x'\in\mathcal{X}} \big\{ f(x',\eta) + \gamma\mathcal{V}(x',\eta) : B(\eta)x + A(\eta)x' = b(\eta) \big\}. \end{aligned} \tag{4.29}$$

The corresponding mapping

$$\mathfrak{T}_\eta(g)(x) := \rho_{|\eta}\left[\Psi_g(x,\eta')\right]$$

depends on η.

For example, consider the risk neutral case of stationary homogeneous Markov chain with the state space $\{(B^1,A^1,b^1),...,(B^k,A^k,b^k)\}$ and probability of moving from state (B^i,A^i,b^i) to state (B^j,A^j,b^j) is p_{ij}. Then the WB equations take the form

$$\begin{aligned} V^i(x) &= \min_{x'\in\mathcal{X}} \big\{ f^i(x') + \gamma\mathcal{V}^i(x') : B^i x + A^i x' = b^i \big\}, \\ \mathcal{V}^i(x) &= \textstyle\sum_{j=1}^k p_{ij} V^j(x),\ i = 1,...,k. \end{aligned} \tag{4.30}$$

Here the value function $\mathcal{V}^i(\cdot)$ should be computed for every $i = 1,...,k$.

To construct stationary homogeneous Markov chain approximation of the true process, we follow the techniques introduced in section 2.2.3. Recall that we search for a partition that minimizes the expected Euclidean distance from the approximating Markov chain and

the true process, given by the objective (2.8). In the stationary setting, we restrict the Markov states to be the same among all stages. In other words, the objective (2.8) is replaced by,

$$\min_{\mu_1,\ldots,\mu_K} \mathbb{E}_\eta \left\{ \sum_{t=2}^{T} \min_{k=1,\ldots,K} \|\eta_t - \mu_k\|_2^2 \right\}, \tag{4.31}$$

where the partition in each stage is the same and is given by μ_k for $k = 1, \ldots, K$. The choice of T is depending on specific applications.

4.8 Using MSPPy to implement the periodical SDDP/SDDiP

In this section we illustrate how to use the MSPPy package to implement the periodical SDDP or periodical SDDiP. The usage syntax is almost identical to sections 3.2 and 3.3. Therefore, we will take the periodical SDDP as an example to illustrate its difference in implementations compared to the classical SDDP. The obvious difference is that we need to call PSDDP/PSDDiP solver instead of SDDP/SDDiP.

Another difference is that we need to specify the number of stages more carefully. Different from the classical SDDP, we can specify different numbers of stages in the steps of solving MSLP demonstrated in section 3.2. In step one, where we build the true problem, the number of stages should be the length of a single period plus one (i.e., m+1). Cutting planes are then added to these m+1 stages in backward passes. In step three, where we solve the discretized problem, we can specify a different number of stages to generate trial points in forward passes. In step four, where we evaluate the obtained policy on the true problem, we could specify another number of stages for evaluation.

For example, the following snippet specifies a 3-stage problem with a period of 2. The backward passes will add cuts to these 3 stages. The forward passes will generate trial points from the first 12 stages. After running the SDDP algorithm, the obtained policy is evaluated up to 60 stages.

```
example = MSLP(T=3)
```

```
...
PSDDP(example).solve(..., forward_T=12)
EvaluationTrue(example).run(..., query_T=60)
```

CHAPTER 5
APPLICATIONS

In this chapter, we apply methodologies and algorithms we have developed in previous chapters to solve various real-world large-scale problems in power system, airline and finance using the MSPPy package.

The hydro-thermal power system planning example in section 5.1 is originated from [40]. This problem has been studied in several publications and has produced many interesting results, for example [40], [41], [42], [43]. We will explore the problem deeper and consider a mix-integer version of the problem which has not been solved in the literature.

The second application in section 5.2 is taken from [44] in which the true demands are modeled as Markovian processes. In that paper, a scenario tree approximation of the demand processes is constructed and used to solve the problem. In the following publication [19], the problem is approximated by making stage-wise independent samplings from the true process and the corresponding Markovian structure is completely ignored. Anyways, neither of these approaches solve the true problem, and the suggested solutions may have a considerable bias. We will show that by using our methodology, the problem can be solved and understood better.

The third application in section 5.3 is a multi-period portfolio optimization problem originated from [45]. In that paper, a three-stage problem with a finite stage-wise independent return process is analyzed. We will illustrate by virtue of the MSPPy package, a more sophisticated/realistic return process can be incorporated and analyzed. In addition, risk management can be easily taken into account.

The fourth application in section 5.4 is motivated from [46]. In that paper, a stochastic optimization model was built to hedge European options. We formulate a similar problem and utilize our methodology to solve it.

In section 5.5, we compare the MSPPy package with other existing open-source software packages. In particular, SDDP.jl is used as the benchmark of performance for the hydro-thermal power system problem. For the ease of comparison, we consider a simple stage-wise independent finite discrete version of the problem.

Meanwhile, we empirically investigate some numerical aspects of multistage stochastic programming. In particular, we find clues of the following questions. How do various discretization techniques for the true process compare to each other? How does the proposed periodical SDDP work, compared to the classical one? How does the SDDP algorithm scale up with increase of the dimension of the state space? Does regularization help? What is the value of the multistage optimization, i.e., why don't we just run the rolling horizon based algorithm instead of taking all the future into account? There are no definite answers to these questions, but we hope through numerical experiments, it will bring some insights and understanding about these questions.

5.1 Hydro-thermal power system planning

5.1.1 Introduction

The Brazilian interconnected power system [40] have four regions $(i = 1, 2, 3, 4)$. In each region i, demand is satisfied by energy generation q_i (in megawatt, the same below) from an integrated reservoir, energy generation g_k from local thermal plants $k \in \Omega_i$ and energy inflow $\text{ex}_{j\to i}$ from other regions ($j \in \{1, 2, 3, 4\}$) or a transshipment station ($j = 5$). If demand d_i can not be satisfied, system i borrows df_{ij} units of energy from the deficit account in system j and a cost of e_{ij} (in dollars per megawatt, the same below) will be incurred. The deficit accounts are set up in a way that ensures the complete recourse condition is held. There are tiny costs for each unit of energy spillage and energy exchanges, denoted by b_i, $c_{j\to i}$ (energy flowed from region j to region i) respectively. We assume a cost of u_k for each unit of energy thermal plant k produces. The dynamics of the system is that the stored energy v_{it} of each reservoir i in stage t is equal to the previous stored energy $v_{i,t-1}$

plus the current water inflow a_{it} minus the energy generation q_{it} and the water spillage s_{it}. The four stored energy $v_{i,t}, i = 1, \ldots, 4$ represents the state variables and the rest are the control variables. The transshipment station is solely for water exchanges between regions. It is required to make decisions sequentially of how much energy to store after seeing the realization of inflow energy as shown in the below chain relationship. Here v_i and a_i are four dimensional vectors. v_0 is given and a_1 is deterministic. The objective is to obtain a sequential scheme to minimize the total operation cost over a design period of T stages with a discount factor γ while meeting energy requirements and feasibility constraints. We add additional subscripts t to distinguish variables and costs at different stages,

$$v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \ldots$$

$$\begin{aligned}
\min \sum_{t=1}^{T} \gamma^{t-1} \Bigg\{ & \sum_{i=1}^{4} \Big[b_{it} s_{it} + \sum_{j=1}^{5} \mathrm{c}_{j\to i,t} \mathrm{ex}_{j\to i,t} + \sum_{k\in\Omega_i} u_k g_{kt} + \sum_{j=1}^{4} e_{ij} \mathrm{df}_{ijt} \Big] \\
& + \sum_{j=1}^{4} c_{j\to 5,t} \mathrm{ex}_{j\to 5,t} \Bigg\}
\end{aligned}$$

s.t. $\forall t = 1, \ldots, T$

$$v_{it} + s_{it} + q_{it} - v_{i,t-1} = a_{it}, \forall i,$$

$$q_{it} + \sum_{k\in\Omega_i} \mathrm{g}_{kt} + \sum_{j=1}^{4} \mathrm{df}_{ijt} - \sum_{j=1}^{5} \mathrm{ex}_{i\to j,t} + \sum_{j=1}^{5} \mathrm{ex}_{j\to i,t} = d_{it}, \forall i,$$

$$\sum_{j=1}^{4} \mathrm{ex}_{j\to 5,t} = \sum_{j=1}^{4} \mathrm{ex}_{5\to j,t},$$

$$0 \le v_{it} \le \bar{v}, 0 \le q_{it} \le \bar{q}_i, \forall i,$$

$$\underline{g}_k \le g_{kt} \le \bar{g}_k, \forall k,$$

$$0 \le df_{ijt} \le \bar{df}_{ij}, 0 \le ex_{i\to j,t} \le \bar{ex}_{i\to j}, \forall i, \forall j,$$

$v_{i,0}$ are given, $a_{i,1}$ are deterministic, $\forall i$.

The problem is originally a 60-stage (60-month) multistage problem with a monthly discount factor $\gamma = 0.9906$ (this discount factor corresponds to the annual discount rate of 12%, i.e., $1/\gamma^{12} = 1.12$ for $\gamma = 0.9906$). In order to deal with the so-called end-of-horizon effect, 60 more stages are added to the problem, which results in $T = 120$ stage problem. The inflow energy often shows a seasonal pattern and is thus natural to be modeled by time series. The immediate question is how to incorporate the time series equations into the optimization problem and make discretization. As discussed in chapter 2, it can be done by either a time series approach (TS) or Markov chain approximation (MCA). We show the formulation implied by these two approaches specifically for this problem in section 5.1.3 and section 5.1.4 respectively. In section 5.1.5, we implement these two approaches by the classical SDDP algorithm and compare them. In section 5.1.6, a thermal security constraint is added to the problem and makes the problem mix-integer. We use SDDiP to solve the problem for tiny test problems in which we intentionally set the number of stages to be small to see how the algorithm performs. The reason for setting a small number of stages is simply because it is too computational expensive to solve the original mix-integer problem. Alternatively, from sections 5.1.7, we discuss the periodical approach that does not suffer from the end-of-horizon effect at all and may significantly reduce the time complexity in settings including risk neutral, risk averse and mix-integer. More importantly, this is the first time that the mixed-integer version of the problem is solve in the literature. Sections 5.1.8 and 5.1.9 are dedicated to the regularized SDDP algorithm and the rolling horizon based algorithm respectively. In section 5.1.10, we artificially increase the dimension of the problem and show the performance of SDDP and its regularized variant. All experiments in this section were done using MSPPy [1], Gurobi 8.1.0, Python 3.7.1 in Red Hat Linux.

5.1.2 Modeling the inflow energy

As shown in [40] and [43], the right hand side energy inflow process $\{a_{it}\}$ can be modeled as an autoregressive process. Let X_t denote the data process of the four dimensional inflow energy. After log-transformation (taken component-wise) $Y_t = \log(X_t)$ of the original data process X_t, a periodical first order autoregressive model is fitted to Y_t with period $m = 12$,

$$Y_t - \mu_t = \Phi_t(Y_{t-1} - \mu_{t-1}) + \epsilon_t$$

Where $\mu_{t+12} = \mu_t$ are the four-dimensional monthly averages of Y_t, $\Phi_{t+12} = \Phi_t$ are four by four coefficient matrices, and ϵ_t are four-dimensional independent errors with periodical multivariate normal distribution $N(0, \Sigma_t)$ with $\Sigma_{t+12} = \Sigma_t$.

The above model is not linear with respect to X_t. Plug in $Y_t = \log(X_t)$ and linearize the model through first order Taylor expansion approximation, we get the following autoregressive process with a multiplicative error,

$$X_t = e^{\epsilon_t} \circ \left[\Phi_t e^{\mu_t - \mu_{t-1}} \circ X_{t-1} + (I - \Phi_t)e^{\mu_t}\right], \tag{5.1}$$

where $\circ$ denotes the Hadamard product and I denotes the four by four identity matrix. The exponential function $e^{(\cdot)}$ is understood component-wise. We will fit this model to data.

Let $R_t = e^{-\mu_t} \circ (X_t - e^{\mu_t})$ and set ϵ_t to 0, the above model can be written as

$$R_t = \Phi_t R_{t-1}$$

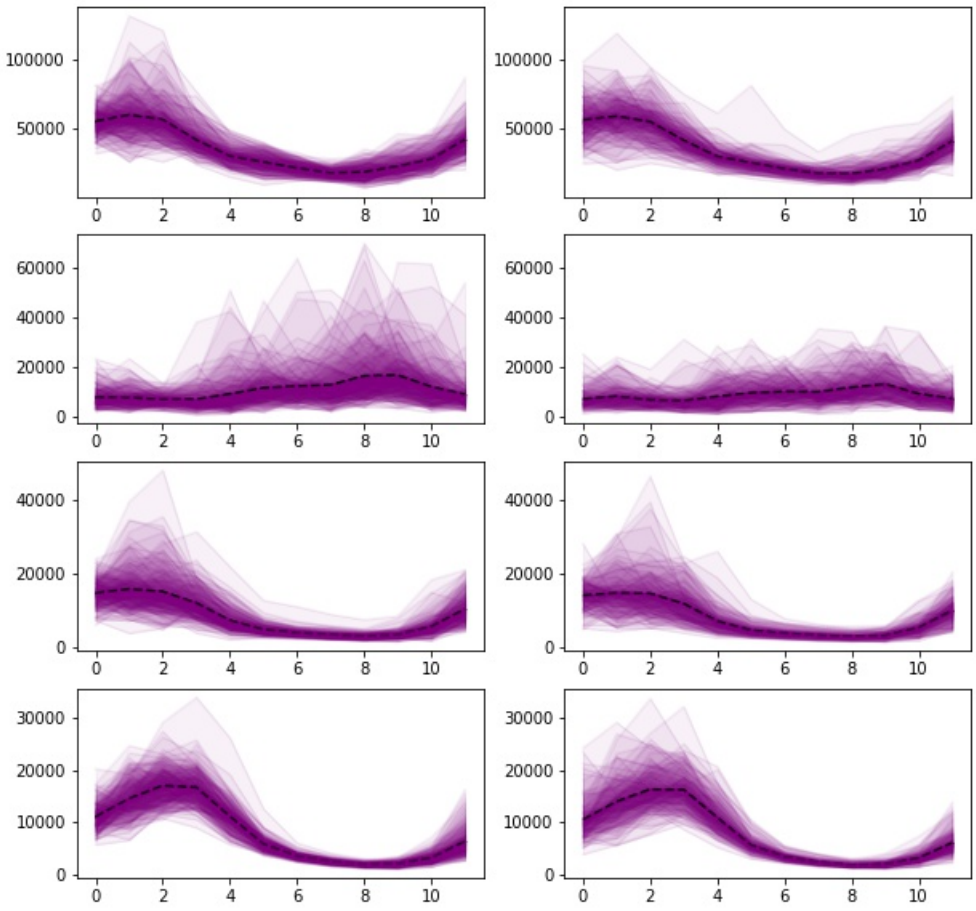
Φ_t is then calibrated by least squares and Σ_t is modeled as sample covariance matrix for

$$\log(X_t) - \log\left((I - \Phi_t)e^{\mu_t} - \Phi_t e^{\mu_t - \mu_{t-1}} \circ X_{t-1}\right),$$

where $\log(\cdot)$ and $e^{(\cdot)}$ are again understood component-wise. Statistical validation of such a

model is discussed in [40]. Figure 5.1 visualizes the model fit using the MSPPy package. It is clear that the shapes are similar, except that the simulated data for the second reservoir have a bit more tails than the real data. Overall, the above time series model fits well to the data. From now on, we will regard equation (5.1) as the true data process.

Figure 5.1: The simulated inflow using fitted model (on the left) and the historical data (on the right). The black dash line represents the average value.



5.1.3 Time series approach (TS)

In the time series approach, the true process (5.1) is included by regarding the inflow energy as additional state variables. The uncertainty in the problem becomes the right hand side white noise in (5.1) and thus, the problem is transformed to be stage-wise independent. As a result, the classical SDDP algorithm and analysis becomes available. But the disadvantage is also clear: the dimension of the state space is doubled.

5.1.4 Markov chain approximation (MCA)

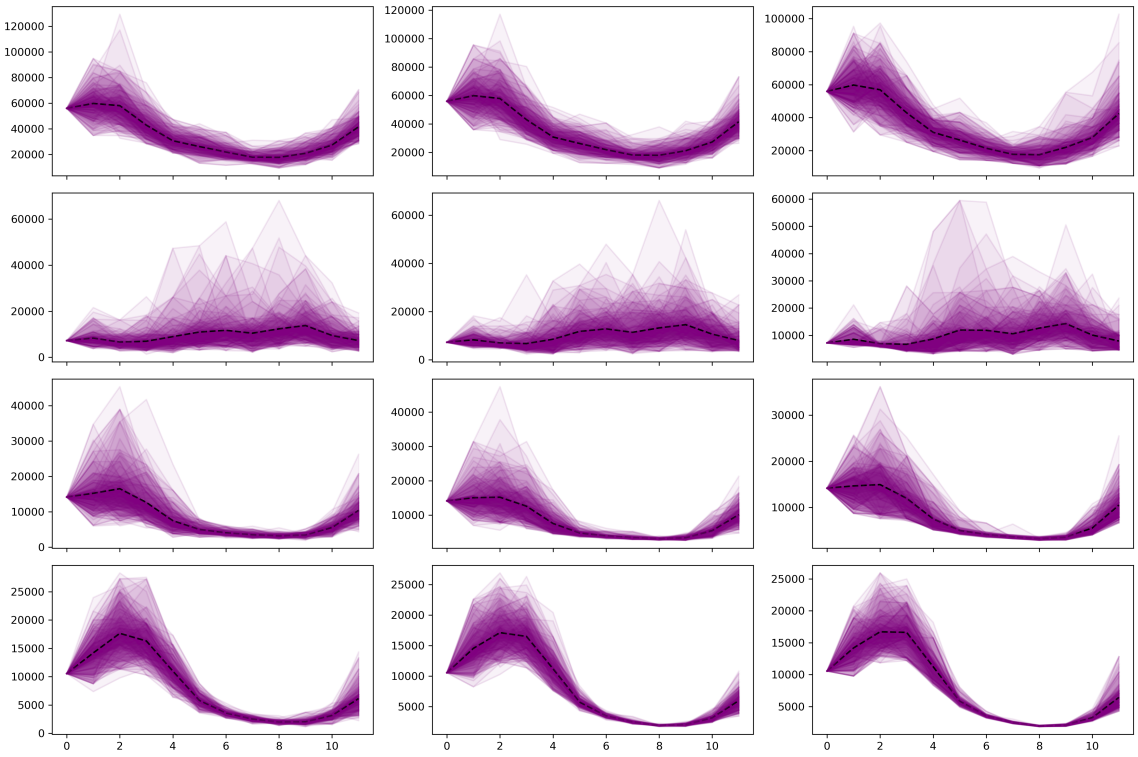
In the Markovian approach, the true process is not added directly to the original problem. Instead, the true process is approximated by Markov chain approximation. Figure 5.2 visualizes the true process and its Markov chain approximation using the MSPPy package. This approach has a benefit of keeping the state space untouched. But it increases the space complexity multiple times and adds an additional layer of approximation (see section 2.2.3).

5.1.5 Comparing TS with MCA

In Table 5.1, we provide a summary of the construction and solving of the discretized problems. The first column gives the discretization methods and granularity of discretization. Columns 2 and 3 show the number of sample paths and the time cost to train the approximating Markov chain. Columns 4 and 5 report the number of iterations and the time cost for performing SDDP algorithm. Column 6 reports the total time in getting the policies. The last two columns report the statistical gaps and deterministic bounds. It manifests that SA/RSA/SAA policies consume much longer time per iteration to obtain than TS policies. Part of the reasons is due to the high memory consumption of the Markov chain approach. On the other hand, SA/RSA/SAA policies produce much smaller gaps.

In particular, Figure 5.3 visually compares the evolution of gaps for SA100 and TS100. The approximate confidence intervals for the expected policy value are computed based

Figure 5.2: The simulated inflow using approximating Markov chain calibrated by SAA (on the left), SA (in the middle), and RSA (on the right)



on the policy values obtained from six forward passes per iteration. The statistical upper bounds are then the upper end of the approximate confidence intervals. To better depict the trend of the statistical upper bounds, the mean value of the policy values obtained from the six forward passes U_i for each iteration $i = 1, \ldots, N$ is smoothed by a convex curve $x_i, i = 1, \ldots, N$ that solves the following quadratic program proposed in [43],

$$\min_{x_1,\ldots,x_N} \left\{ \sum_{i=1}^{N} (x_i - U_i)^2 : x_i + x_{i-2} \geq 2x_{i-1}, x_{i-1} \geq x_i \right\}$$

In Table 5.2, we pairwisely compare the obtained seven policies with 3000 Monte Carlo simulations. In Table 5.3 we provide 95% confidence intervals for both the discretized and true problems. We make the following observations. First, the granularity of discretization does matter. In this case, 50 samples per stage are not sufficient for both the TS approach and MCA. Second, given the same granularity of discretization, TS policies dominates the

Table 5.1: SDDP implementation results for discretized problems

model	#iter. MCA(M)	MCA (sec.)	#iter. SDDP	SDDP (sec.)	total (sec.)	gap	bound ($M)
SA50	0.1	300	100	2,163	2,463	11.6%	185.0
SA100	1	4,039	100	4,771	8,810	13.0%	186.7
RSA50	0.1	346	100	2,142	2,488	10.7%	181.3
RSA100	1	4,513	100	4,829	9.342	12.6%	184.2
SAA50	0.1	10,521	100	2,122	12,643	11.0%	177.2
TS50	/	/	293	2,412	2,412	27.0%	186.2
TS100	/	/	400	7,724	7,724	23.3%	188.6

Figure 5.3: Comparison of SA100 (on the left) and TS100 (on the right)

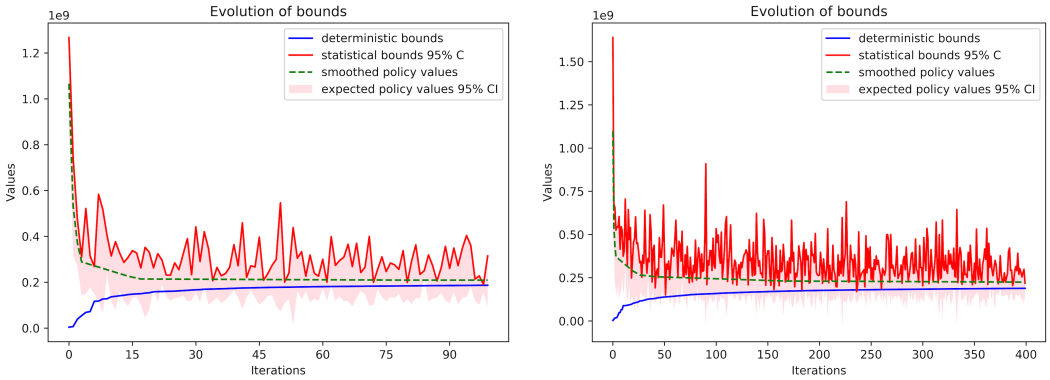


Table 5.2: One-sided p-values for pairwise comparison of policies. P-values are written in the lower triangular part if the sample mean value of the row policy is lower than that of the column policy (i.e., the row policy is better than the column policy).

	SA50	SA100	RSA50	RSA100	SAA50	TS50	TS100
SA50			<0.001	<0.001	<0.001		
SA100	<0.001		<0.001	<0.001	<0.001		
RSA50					<0.001		
RSA100			<0.001		<0.001		
SAA50							
TS50	0.127	0.249	0.022	0.083	0.002		
TS100	0.025	0.068	0.002	0.013	<0.001	<0.001	

MCA policies, although TS policies are obtained by solving the discretized problem with a larger optimality gap. Third, the SA approach for MCA seems to dominate the RSA and SAA approaches.

Table 5.3: Policy evaluation

policy	CI for approx. model ($M)		CI for true problem ($M)	
SA50	201.1	206.5	228.3	236.5
SA100	205.2	211.0	227.0	235.1
RSA50	195.3	200.8	230.8	239.1
RSA100	201.4	207.3	229.1	237.1
SAA50	191.6	196.8	233.6	242.1
TS50	228.8	236.4	225.3	233.0
TS100	225.1	232.5	223.1	230.6

5.1.6 A thermal security constraint

One of the concerns about the Brazilian power system is to include a thermal security constraint. That is, whenever the stored energy is below certain level, thermal energy must be produced to ensure enough operational maneuver options for hydro plants and for electrical network reliability. This can be achieved by including binary variables as follows,

$$v_{it} \geq z_i v_i^*, \sum_{k \in \Omega_i} g_{kt} \geq (1 - z_i) g_i^*, g_i^* \text{ and } v_i^* \text{ given}, z_i \in \{0, 1\}, \forall i. \tag{5.2}$$

Adding the above additional binary variables and constraints makes the problem a multistage stochastic mixed-integer program. Such problems are solved by the so-called SDDiP algorithm introduced in section 2.8. In this section, we make some preliminary attempts to solve the above problem by SDDiP with strengthen Benders' cuts for only up to 6 stages. In section 5.1.7 we will see an alternative approach to make this problem computationally more tractable.

Table 5.4 summarizes solutions to the problem when the number of periods (stages) is small. In column 4, both of the extensive solver and the SDDiP solver provide lower bounds for the discretized problems. In column 5 for $T = 2, 3$, both of the solvers provide upper bounds for the discretized problems; more specifically, the extensive solver gives the best-so-far objective found in branch and bound; the SDDiP solver gives the exact expected policy values by exhaustively evaluating the policy under all scenarios (this is

doable for tiny problems since the number of total scenarios is small). In column 5 for $T = 6$, the SDDiP solver provides a confidence interval of the expected policy values with 3000 Monte Carlo simulations. All discretized problems are constructed by the TS approach with 100 i.i.d. samples for each stage. Observe that the extensive solver quickly becomes obsolete with an increasing number of stages. But it still validates the correctness of the SDDiP solver when the number of stages is small (the number of total scenarios is 10^2 for $T = 2$ and 100^3 for $T = 3$). Comparing the solutions given by different types of cuts, Strengthened Benders' cut and Lagrangian cut are able to reduce the optimality gap (column 7) dramatically for discretized problems. Surprisingly, as shown in the last column, their benefits to the true problem are not observable in this case.

Table 5.4: Summary of solving a mix-integer power system problem

solver	cut	T	LB	UB/ CI appr.	#iter. SDDiP	gap	time (sec.)	CI true
Ext.	/	2	1,623,203	1,623,203	/	0.00%	<1	/
SDDiP	B	2	1,623,203	1,623,203	10	0.00%	<1	1,624,384 1,626,363
Ext.	/	3	3,078,162	3,084,143	/	0.19%	1000	/
SDDiP	B	3	2,110,928	3,074,398	303	45.64%	15	3,071,920 3,079,293
SDDiP	SB	3	2,998,418	3,074,398	57	2.53%	15	3,071,920 3,079,293
SDDiP	LG	3	2,999,141	3,074,398	19	2.51%	15	3,071,926 3,079,299
SDDiP	B	6	3,116,866	4,999,105 5,013,830	345	60.86%	60	5,000,001 5,022,540
SDDiP	SB	6	4,946,515	5,002,811 5,056,414	53	2.22%	60	5,004,602 5,081,744
SDDiP	LG	6	4,946,263	5,005,058 5,060,038	33	2.30%	60	5,005,753 5,083,226

5.1.7 Periodical SDDP algorithm

As discussed at the beginning of this chapter, 60 stages are added to the problem to mitigate the end-of-horizon effect. However, implementing the classical SDDP algorithm for the resulting 120-stage problem requires huge computational effort and resources. By contrast, modeling the problem as an infinite horizon periodical problem leverages the periodical structure of the problem and only needs to solve a 13-stage problem (with deterministic first stage problem and $\mathsf{m} = 12$). For the discount factor $\gamma = 0.9906$ in risk neutral setting, the error estimate given in (4.26), is $\gamma^T/(1-\gamma) \approx 34$. This estimate is still large, indicating that the cost-to-go function at the 120th stage may be far from zero and the end-of-horizon effect may still be significant. If setting a smaller discount factor, for example $\gamma = 0.8$, the coefficient in (4.26) becomes $\gamma^T/(1-\gamma) \approx 1.17e-11$, while the upper bound κ of the objective is less than $1e10$ for this problem. It then makes sense to regard the cost-to-go function at the 120th stage as zero. In such a case, the 120-stage problem is approximately the same as the infinite horizon problem. It can be noted that the monthly discount factor $\gamma = 0.8$ may not have an economic sense, but it is still useful for algorithmic comparisons.

In this section, we numerically compare the classical SDDP and the proposed (periodical) variant of the SDDP algorithm on this Brazilian power problem with discount factors 0.8 and 0.9906 in risk neutral, risk averse, and integer settings. The true problem is discretized by SAA with sample size $N_t = 100$ at every stage $t \geq 2$. Both algorithms are implemented on SAA discretized problems for 300 iterations and are parallelized by 10 processes (i.e., 10 forward and backward steps are simultaneously done for each iteration, and hence in total effectively around 3000 iterations are implemented). For every 100 iterations, we evaluate the obtained policy from both approaches and compare the bound, policy value and the solution. Since the periodical SDDP solves a much smaller problem per iteration (ten times smaller in this case since there are around 10 periods for $T = 120$ and $\mathsf{m} = 12$), if the comparison shows similar results, the periodical SDDP will be significantly more efficient for stationary multistage problems than the classical SDDP.

Risk neutral problem

We start with the risk neutral version of the problem, with $\varrho = \mathbb{E}$, and discount factor $\gamma = 0.8$. Table 5.5 compares the speed of convergence for the two approaches. The lower bound, confidence interval of policy values, and optimality gap evolve in a similar pattern for the classical SDDP and periodical SDDP. The periodical SDDP has similar rate of convergence for the same number of iterations. Figure 5.4 illustrates the evolution of individual stage costs both for the discretized problem and the true problem. For both approaches, the individual stage costs stabilize after 300 iterations. Figure 5.5 exhibits the evolution of stored energy in one of the regions for the discretized problem and the true problem. The stored energy stabilizes very quickly and it is slightly more cyclical for the periodical SDDP than for the classical SDDP. Both plots also reflect that the discretized problem is a good approximation to the true problem.

Table 5.5: Comparison of classical SDDP and periodical SDDP for discount 0.8, with the lower bound, the 95% confidence interval of policy values for the SAA discretized problem and the true problem, and the optimality gap for the SAA discretized problem

	periodical SDDP			classical SDDP		
#iter	LB ($m)	CI approx.($m) CI true($m)	gap	LB ($m)	CI approx.($m) CI true($m)	gap
100	7.09	7.57, 8.03 7.32, 7.81	13.27%	7.14	7.70, 8.19 7.42, 7.93	14.71%
200	7.31	7.55, 8.01 7.32, 7.81	9.56%	7.30	7.67, 8.15 7.40, 7.91	11.6%
300	7.39	7.51, 7.97 7.27, 7.76	7.84%	7.37	7.65, 8.13 7.38, 7.89	10.3%

Figure 5.4: Individual stage costs (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral case with discount factor 0.8, after 100, 200 and 300 iterations

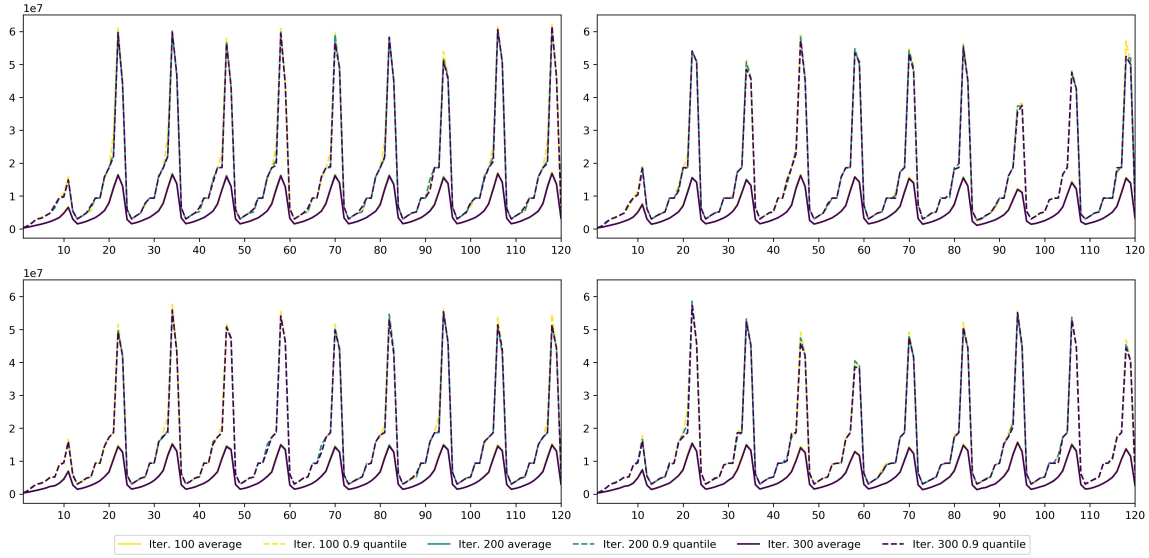
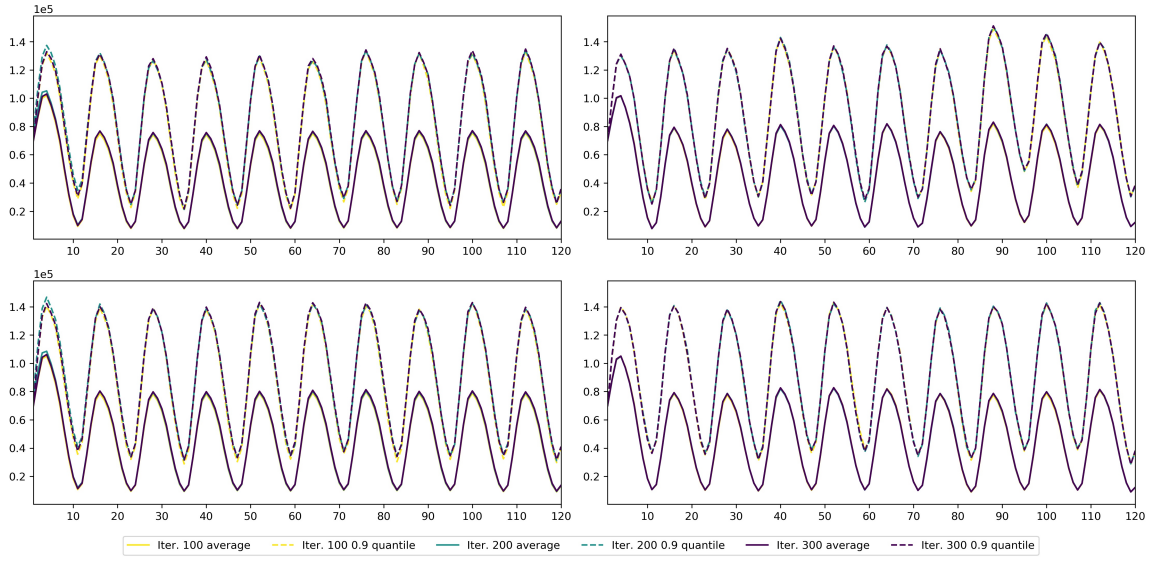


Figure 5.5: Stored energy (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral case with discount factor 0.8, after 100, 200 and 300 iterations



Changing discount factor from 0.8 to 0.9906

The original problem has a discount factor of $\gamma = 0.9906$. As shown above, the 120 stage problem may not be a good approximation of the infinite horizon problem. Indeed as shown in Figure 5.6, the implementation results demonstrate significant discrepancies between the two approaches. The stage costs for the periodical SDDP are similar to those for the classical SDDP for about the first 10 stages, while in the classical SDDP they decay faster in later stages. It appears that the reason for that is the end-of-horizon effect: when approaching the end of the horizon, the classical SDDP tends to use up all available stored energy and thus reduces the stage costs. This argument is verified by Figure 5.7. That being said, in practice, people typically only consider the costs for the first few stages. From that point of view these two approaches behave in a similar way.

Figure 5.6: Individual stage costs (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral case with discount factor 0.9906, after 100, 200 and 300 iterations

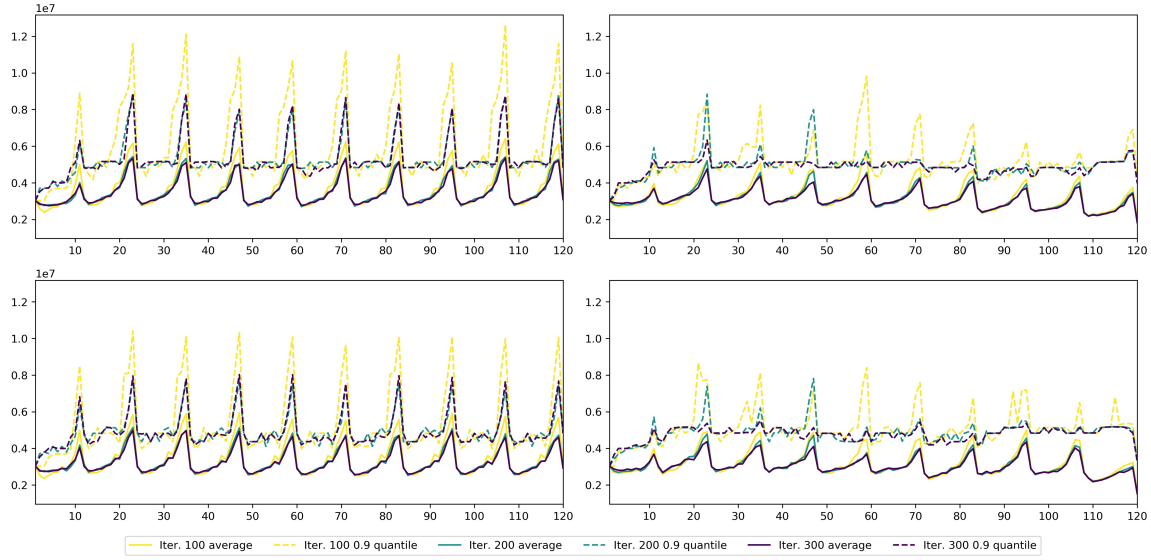
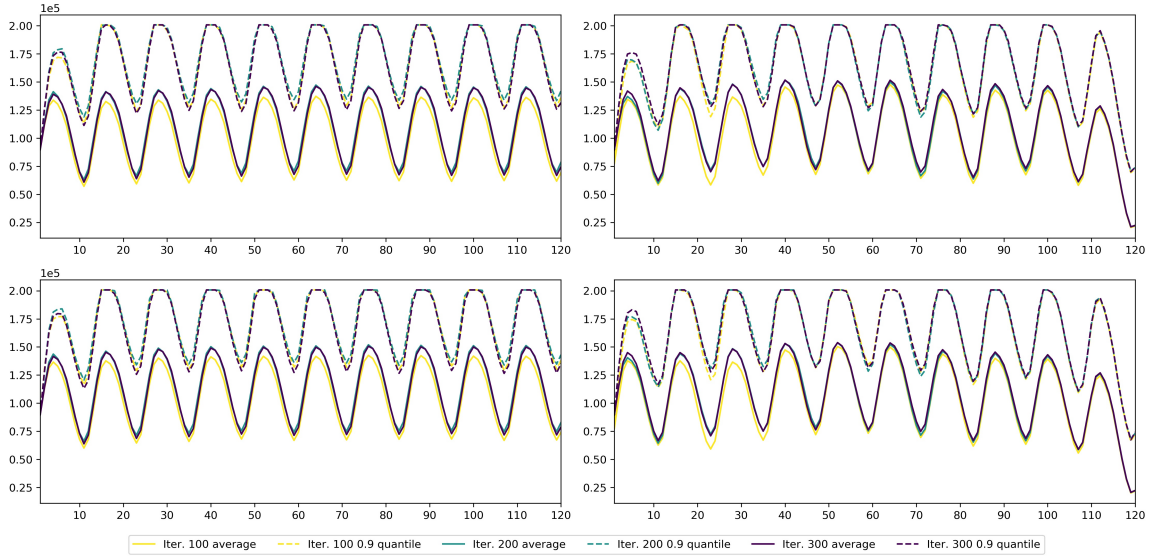


Figure 5.7: Stored energy (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral case with discount factor 0.9906, after 100, 200 and 300 iterations



Markov chain approximation

As we discussed in section 4.7, it is also possible to use the periodical SDDP with Markov chain approximation. Also, as shown in section 5.1.5, the MCA approach, compared to the TS approach, brings in an additional layer of approximation and shows a larger discrepancy between its performance on the discretized problem and the true problem. Therefore, we expect for the periodical SDDP, such discrepancy may even be larger.

We consider the discount factor $\gamma = 0.8$ and set $T = 13$ in the objective (4.31). Table 5.6 shows implementation result of periodical SDDP-MCA (again, here we are using 10 processes). The performance gap between the discretized problem and the true problem is around 10%. Compared to table 5.5, the confidence interval for the true problem is much higher than that obtained by the periodical SDDP-TS.

Table 5.6: Markov chain-periodical SDDP

#iter	LB ($m)	CI approx. ($m) CI true ($m)	gap
100	641.0	682.7, 723.5 780.7, 836.6	12.87%
200	644.0	681.2, 721.8 777.4, 833.1	12.09%
300	645.0	681.5, 722.2 778.0, 833.8	11.97%

Incorporating risk

We compare now the two approaches in a risk averse setting. Specifically, we consider a risk measure which is a convex combination of the expectation and the Average Value-at-Risk,

$$\text{AV@R}_\alpha(Z) := \inf_{x\in\mathbb{R}}\{x + \alpha^{-1}\mathbb{E}[Z-x]_+\},$$

risk measure (also called Conditional Value-at-Risk, Expected Shortfall, Expected Tail Loss),

$$\varrho(\cdot) := (1-\lambda)\mathbb{E}(\cdot) + \lambda\text{AV@R}_\alpha(\cdot),$$

with $\lambda = 0.2$ and $\alpha = 0.1$.

Again, as shown in Figures 5.8 - 5.11, both approaches stabilize after 300 iterations. For discount factor 0.8, the periodical SDDP shows similar but slightly more cyclical result then the classical SDDP. For discount factor 0.9906, the classical SDDP suffers more from the end-of-horizon effect than in the risk neutral setting and the periodical SDDP produces more cyclical and regular solutions.

Figure 5.8: Individual stage costs (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk averse case with discount factor 0.8, after 100, 200 and 300 iterations

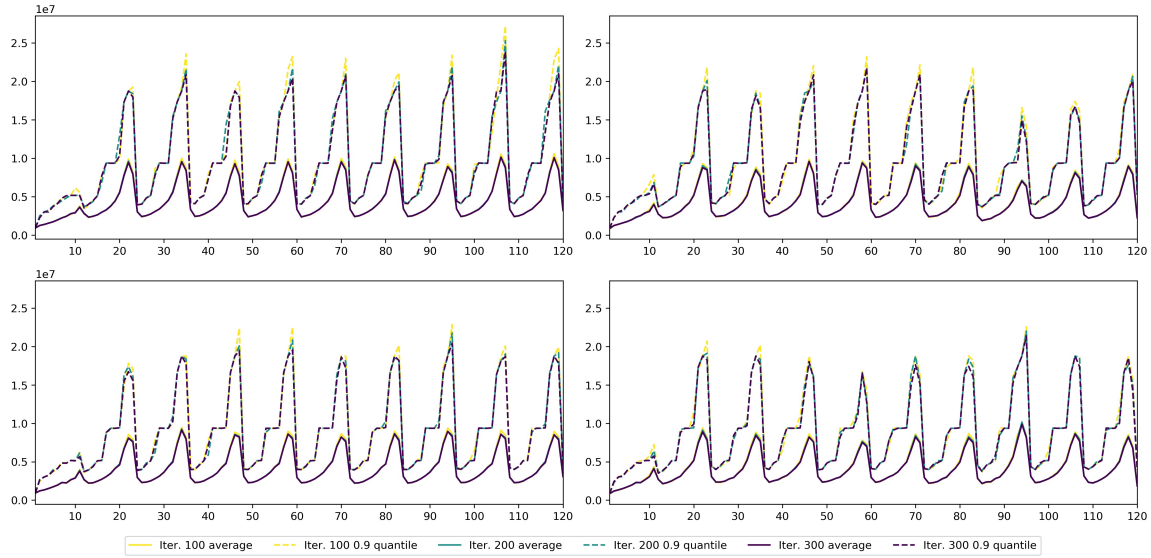


Figure 5.9: Stored energy (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk averse case with discount factor 0.8, after 100, 200 and 300 iterations

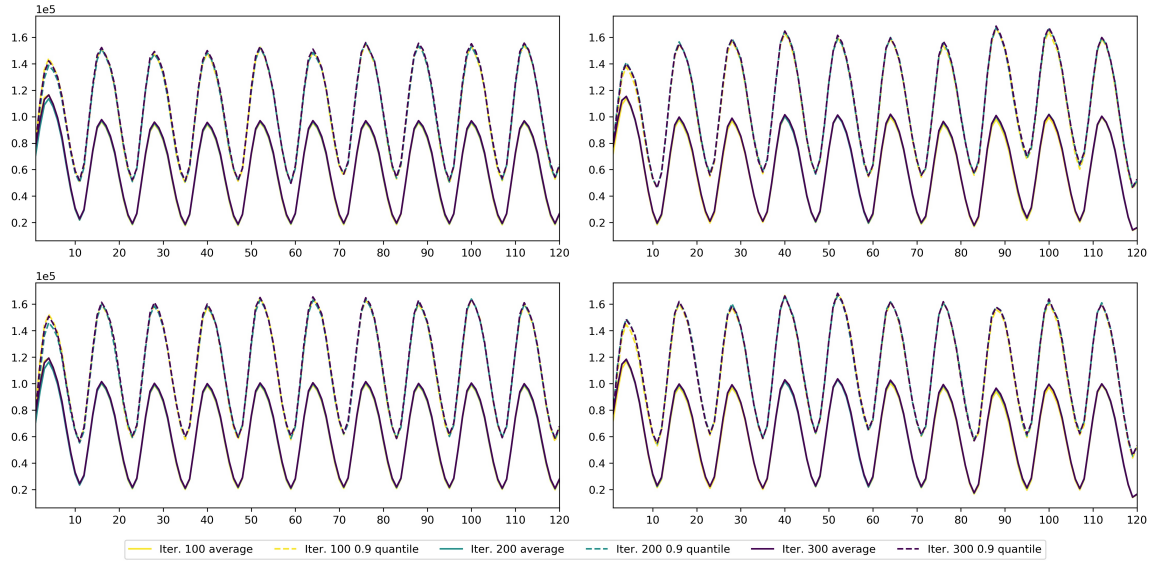


Figure 5.10: Individual stage costs (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk averse case with discount factor 0.9906, after 100, 200 and 300 iterations

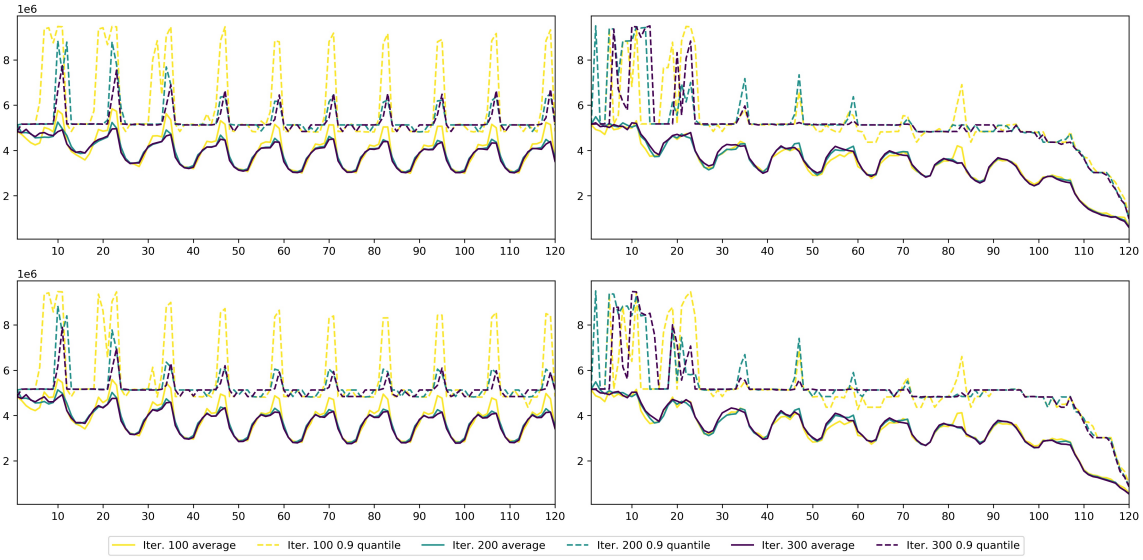
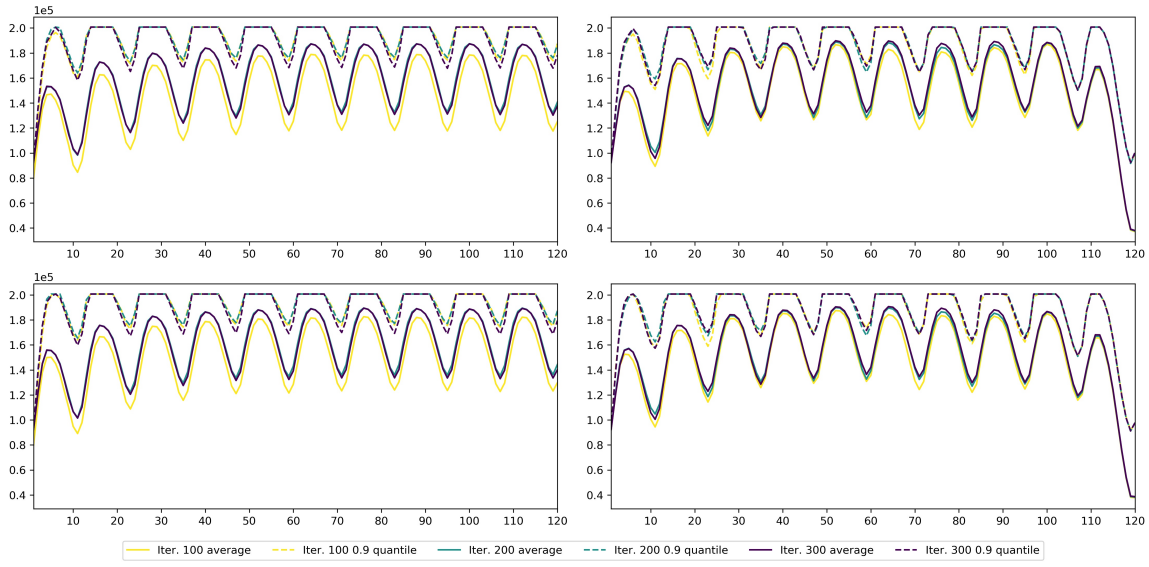


Figure 5.11: Stored energy (in average value and 0.9 quantile) by periodical SDDP (on the left) and classical SDDP (on the right) for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk averse case with discount factor 0.9906, after 100, 200 and 300 iterations



Including binary variables

In section 5.1.6, we introduced the thermal security constraint (5.2) which makes the problem mix-integer. We have used the classical SDDiP to solve the mix-integer problem for only up to 6 stages. When the number of stages become larger, SDDiP is super slow. As far as we know, the problem has not been solved in the literature because of the heavy computational cost.

By virtue of the proposed techniques, we only need to solve the corresponding 13-stage problem, which makes the SDDiP approach computationally much more tractable. As shown in Table 5.7, such large-scale mixed integer problem is able to be solved in a reasonable accuracy. Figures 5.12 and 5.13 compare the results with and without thermal security constraint. For the discount factor 0.9906, the imposed thermal security does not have much impact on the solutions and costs. While for the discount factor 0.8, the imposed thermal security constraint leads to retain more stored energy over the entire horizon.

Figure 5.12: Stored energy (on the left) and discounted stage cost (on the right) in average value and 0.9 quantile for the problem with/without thermal security constraint for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral with discount factor 0.8

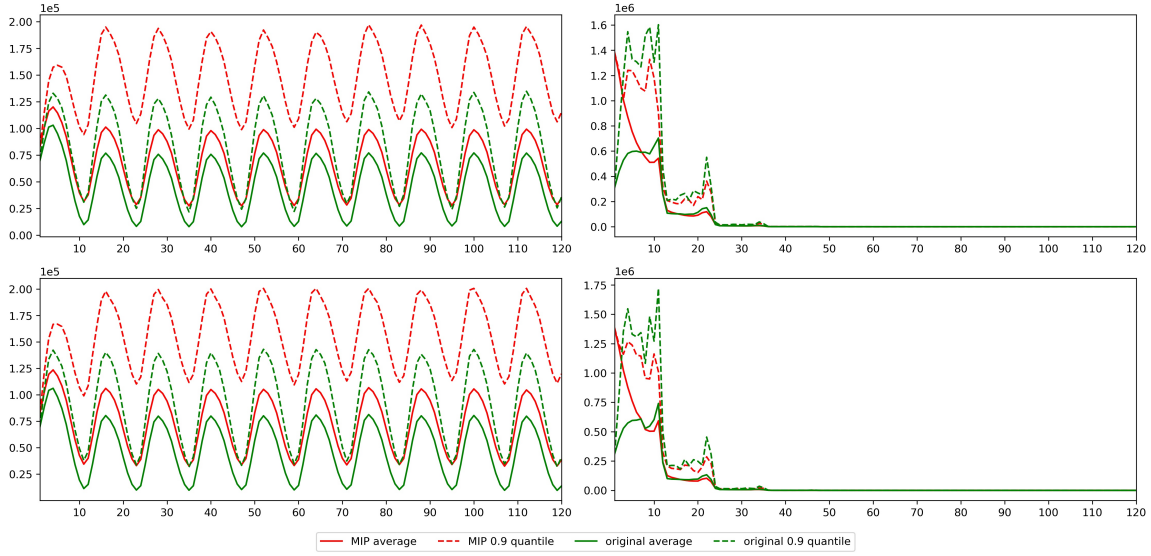


Figure 5.13: Stored energy (on the left) and discounted stage cost (on the right) in average value and 0.9 quantile for the problem with/without thermal security constraint for the SAA discretized problem (on the above) and the true problem (on the bottom) for the risk neutral case with discount factor 0.9906

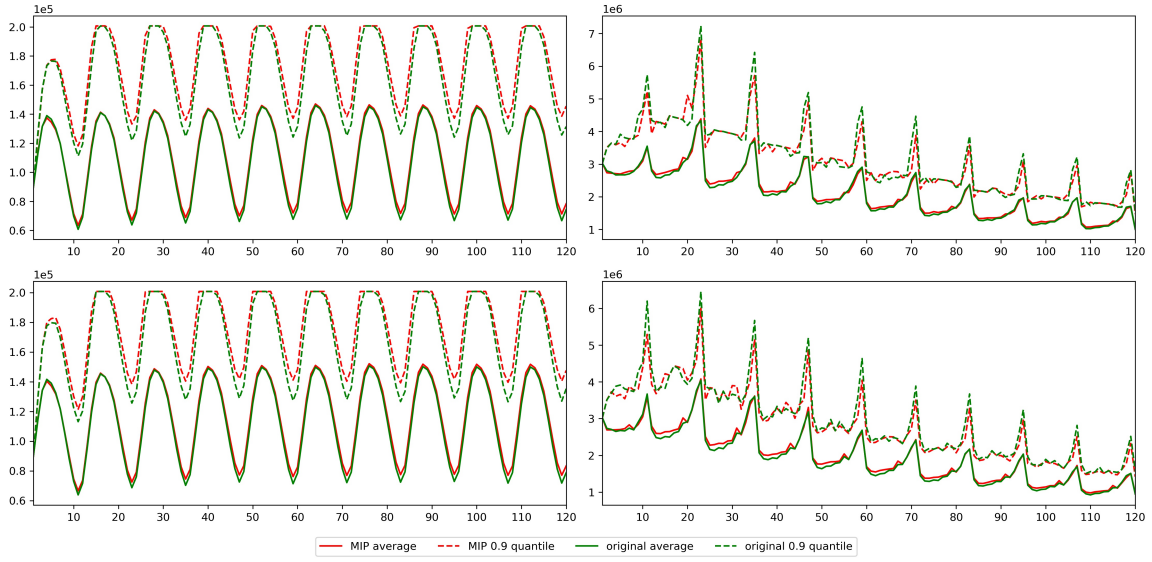


Table 5.7: Result of SDDiP with Benders' cuts for the mixed integer problem with discount factor 0.8.

	periodical SDDP			
#iter	LB ($m)	CI ($m)	CI for true ($m)	gap
100	9.50	9.94, 10.29	9.85, 10.23	8.22%
200	9.65	9.92, 10.26	9.83, 10.21	6.36%
300	9.71	9.90, 10.26	9.84, 10.22	5.67%

Concluding remarks

In conclusion, in all cases above, the proposed periodical SDDP algorithm provides similar, and sometimes more reasonable solutions as opposed to the classical SDDP for the same number of iterations, for *periodical* problems. Giving the benefit of significantly reduced

computational time it brings about, the periodical SDDP algorithm can be an important substitute of the classical one. Having said that, we notice that for the discount factor that is close to 1, it is computationally expensive to evaluate policy value and thus hard to compute the upper bound. We leave it to a future research for construction of a more efficient way to compute the upper bound.

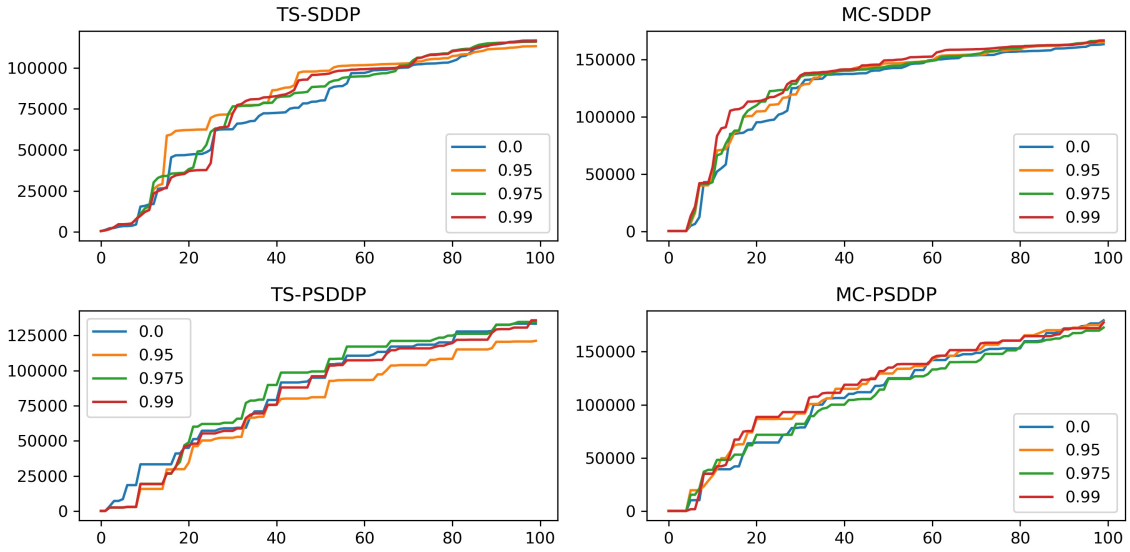
Of course the periodical SDDP algorithm applies only to problems with the periodical structure. It could be mentioned, however, that the user can choose different period lengths m for the considered problem. For instance, in the example of the Brazilian interconnected power system problem, one can use $\mathsf{m} = 24$ (two years) or even $\mathsf{m} = 60$ (five years). Still it will reduce the computational effort and will take care of the end-of-horizon effect.

5.1.8 Regularized SDDP algorithm

We investigate if the regularization approach introduced in section 2.4.4 can boost the convergence rate for the classical SDDP and periodical SDDP. We first point out it is not proper to impose a quadratic penalty directly to the original problem because the penalty terms bring numerical difficulty by making square of the stored energy which is already several thousand (in megawatt). So we rescale the problem to gigawatt (=1000 megawatt). We also ignore the tiny costs of water exchange and hydro power generation. To better demonstrate the effect of regularization, we run the SDDP algorithm serially (i.e., implementing single forward step and backward step for each iteration) and vary the decay factor r to be 0,0.95,0.975 and 0.99, in which $r = 0$ corresponds to the unregularized classical SDDP. The semi-definite matrices S_t are set to be identity matrices.

Figure 5.14 depicts the result for TS-SDDP, Markov chain SDDP, periodical TS-SDDP and periodical Markov chain-SDDP respectively. In general, regularization seems to have no clear benefits in all cases. Though regularized MC-SDDP and MC-PSDDP seems to outperform the unregularized one in early iterations, that advantage quickly vanishes as we move to later iterations.

Figure 5.14: Evolution of lower bounds for 100 iterations by SDDP and regularized SDDP



5.1.9 Rolling horizon algorithm v.s. SDDP algorithm

We compare the rolling horizon algorithm with the classical SDDP algorithm on the original hydro-thermal power system planning problem by varying the number of stages. The continuous data process is discretized by the TS-SAA approach in the classical SDDP algorithm. On the other hand, the rolling horizon algorithm solves the true problem directly.

Table 5.8 shows the implementation result for these two algorithms in five different settings. Column one shows the considered number of stages. Column two records the number of simulations used in computing confidence intervals. Column three shows the granularity of the discretization. For the rolling horizon algorithm it means the number of branches generated in the next stage while for the SDDP algorithm it means the number of i.i.d. samples to construct the SAA discretized problem in each stage. The last three columns provide 95% confidence intervals for the true problems for different random seeds. For the SDDP algorithm, random seeds are used to create SAA discretized problems while for the rolling horizon algorithm they are used to create dynamic scenario trees. We observe that when the number of stages is small, the rolling horizon algorithm is performing well. As the number of stages increases and discretization becomes more granular, the SDDP

algorithm seems to perform much better than the rolling horizon algorithm.

Figures 5.15 to 5.19 show boxplots of individual stage costs in *logarithmic* scale for those five settings. Both algorithms have individual stage costs going down in later stages due to the end-of-horizon effect. One may also expect that the interquartile range (IQR) going down towards the end of the horizon but it is not for both algorithms. This behavior is due to different reasons for these two algorithms. In the rolling horizon algorithm, it is because of its shortsightedness in the sense that in early stages, the construction of its scenario tree does not branch except the next stage and thus it does not take much future into account. While in the SDDP algorithm, it is because the underlying true data process deviates more and more over time from its static scenario tree.

Table 5.8: SDDP v.s. rolling horizon for the power system problem

T	#simulations	#branches	solver	seed 1	seed 2	seed 3
6	500	100	rolling	16.22, 16.88	16.21, 16.88	16.21, 16.88
			SDDP	16.44, 17.20	16.44, 17.22	16.44, 17.20
24	500	10	rolling	53.39, 62.31	53.06, 61.79	53.07, 61.72
			SDDP	55.85, 65.13	57.34, 67.32	58.15, 68.28
	80	100	rolling	49.78, 72.99	49.96, 73.95	49.71, 73.38
			SDDP	42.63, 62.48	42.59, 62.34	42.45, 62.93
120	100	5	rolling	228.7, 276.3	234.5, 284.5	231.2, 279.2
			SDDP	212.7, 247.5	217.2, 262.2	214.5, 259.7
	40	10	rolling	214.0, 303.7	213.1, 304.8	212.1, 305.7
			SDDP	185.4, 243.3	188.9, 262.0	184.5, 241.3

Figure 5.15: Boxplots of individual stage costs in log scale by rolling (on the left) and SDDP (on the right) for a 6-stage problem

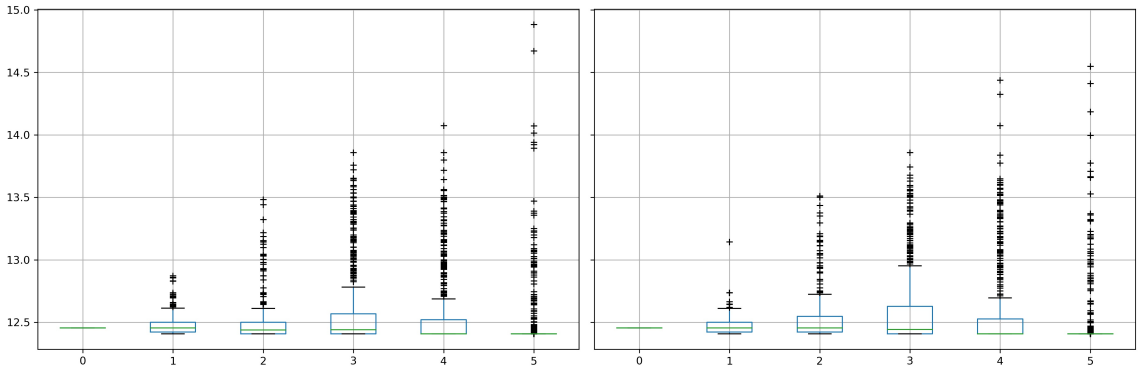


Figure 5.16: Boxplots of individual stage costs by rolling (on the left) and SDDP (on the right) for a 24-stage problem with 10 branches

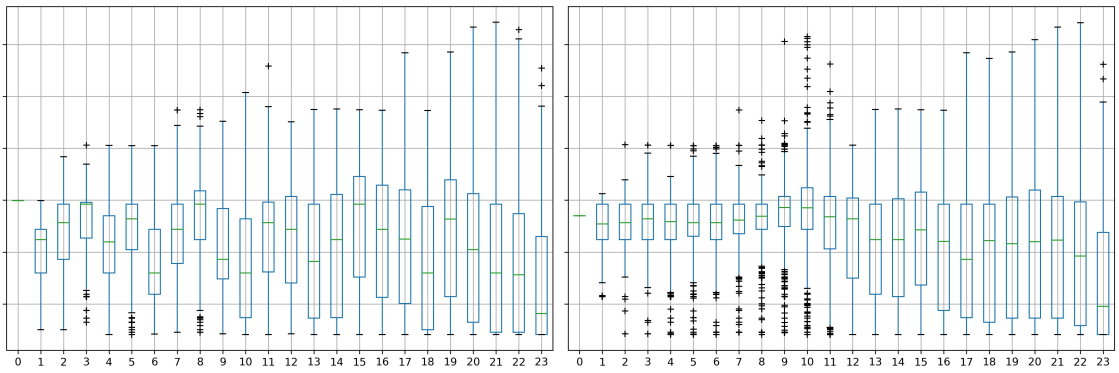


Figure 5.17: Boxplots of individual stage costs by rolling (on the left) and SDDP (on the right) for a 24-stage problem with 100 branches

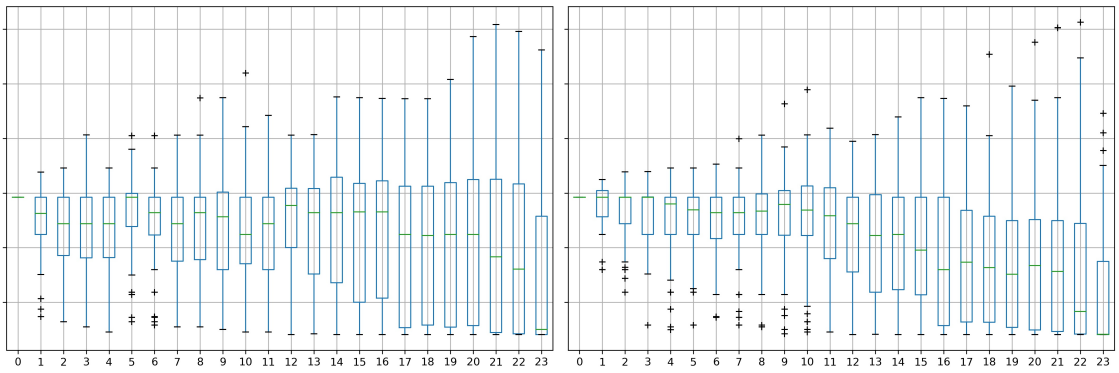


Figure 5.18: Boxplots of individual stage costs by rolling (on the left) and SDDP (on the right) for a 120-stage problem with 5 branches

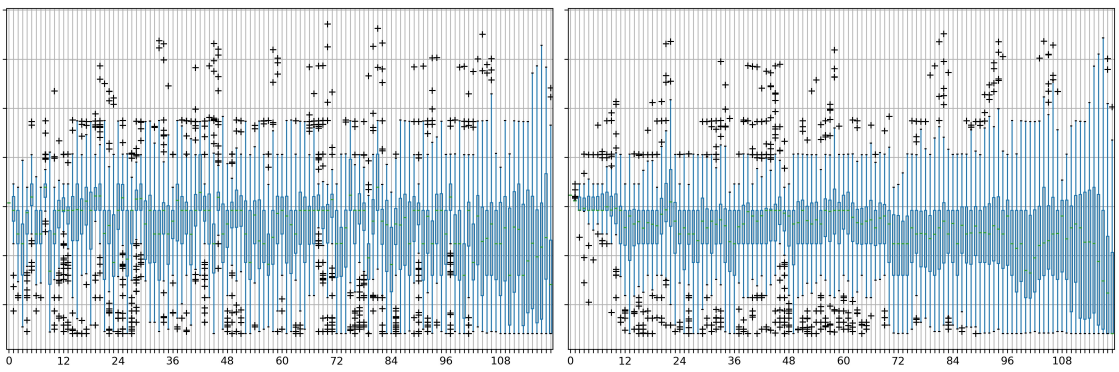
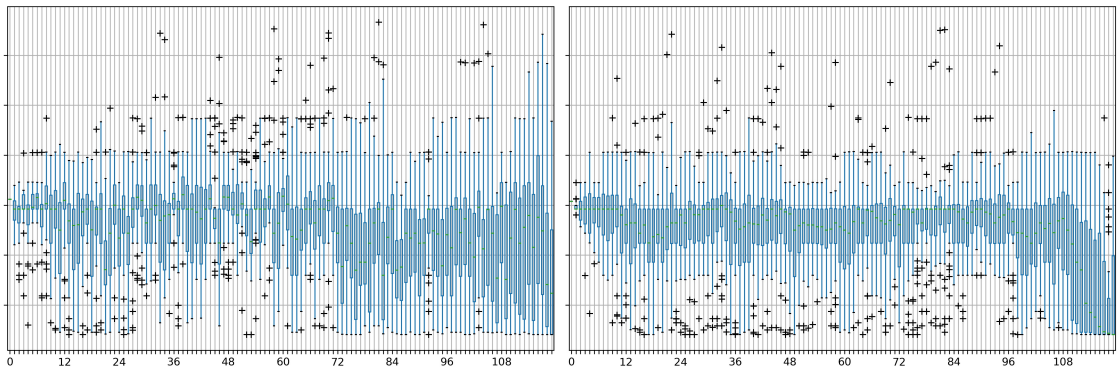


Figure 5.19: Boxplots of individual stage costs by rolling (on the left) and SDDP (on the right) for a 120-stage problem with 10 branches



5.1.10 Increasing the number of reservoirs

In this section, we artificially augment the dimension of the data to investigate how the SDDP algorithm scales up with increase of the number of state variables.

We create an eight-dimensional problem by doubling the existing system (composed of four subsystems and one transshipment station) to two systems. We first assume there is no water exchange between the two systems so that the two systems are completely separate. Recall that we have modeled the original four-dimensional inflow energy as a periodical auto-regressive process with multiplicative four-dimensional independent errors following periodical multivariate log-normal distribution $\log \mathcal{N}(0, \Sigma_t)$. After augmenting the data, we let the resulting eight dimensional independent errors follow $\log \mathcal{N}(0, \hat{\Sigma}_t)$, where

$$\hat{\Sigma}_t = \begin{pmatrix} \Sigma_t & \rho\Sigma_t \\ \rho\Sigma_t & \Sigma_t \end{pmatrix}$$

We implement the periodical SDDP algorithm parallelized by 20 processes on this eight dimensional problem with continuous data process discretized by the TS-SAA approach. The discount factor is set to be 0.8. We vary the value of correlation ρ to see how it influences the convergence. Table 5.9 shows the implementation results. In all implemen-

tations, we keep the sample size as 100 per stage, which might not be enough for this new sixteen-dimensional problem. Indeed, as shown in the tables below, we can see a moderate difference between confidence interval for the discretized problem and confidence interval for the true problem. Nevertheless, it is already clear that the SDDP algorithm does not perform well even on the discretized problem, not to mention its performance on the true problem. Especially when the correlation of the data process is low, the optimality gap for the discretized problem is much larger than the corresponding gap for the original 4-dimensional problem.

Table 5.9: Results for solving a sixteen-dimensional power system problem

ρ	#iter	LB ($m)	CI ($m)	CI for true ($m)	gap
1	100	13.24	13.92, 14.83	14.66, 15.64	12.00%
	200	13.56	13.84, 14.74	14.58, 15.56	8.70%
	300	13.69	13.81, 14.71	14.55, 15.52	7.45%
0.75	100	13.01	16.00, 16.92	15.36, 16.31	30.00%
	200	13.63	15.85, 16.75	15.25, 16.19	22.86%
	300	13.94	15.76, 16.66	15.15, 16.09	19.50%
0.5	100	11.49	15.06, 15.87	15.37, 16.25	38.14%
	200	12.24	14.91, 15.71	15.24, 16.11	28.35%
	300	12.59	14.85, 15.64	15.17, 16.03	24.28%
0.25	100	11.25	15.29, 16.02	15.53, 16.35	42.33%
	200	12.11	15.16, 15.88	15.37, 16.18	31.14%
	300	12.49	15.08, 15.80	15.34, 16.14	26.50%
0	100	11.08	14.95, 15.67	15.31, 16.03	41.35%
	200	11.96	14.76, 15.70	15.15, 15.86	29.32%
	300	12.37	14.88, 15.58	15.08, 15.78	24.63%

We now visualize the dimension of the obtained policies when the correlation $\rho = 0$ or $\rho = 1$. We achieve that by running 3000 Monte Carlo simulations, obtain solutions implied by the policy, and implement principle component analysis on the solutions. The dimension of the obtained policy not necessarily implies the dimension of the optimal policy but it to some extent represents the dimension of the space where the SDDP algorithm is searching for the best policy. Intuitively, if the dimension of the search space is too high, algorithms will have difficulty finding optimality. Indeed, as shown in figures 5.20 and 5.21 where we

show the explained ratio by the principle components, the dimension of the search space for $\rho = 0$ is around 10 while that for $\rho = 1$ is around 5.

Figure 5.20: Dimension of the obtained policy when $\rho = 0$

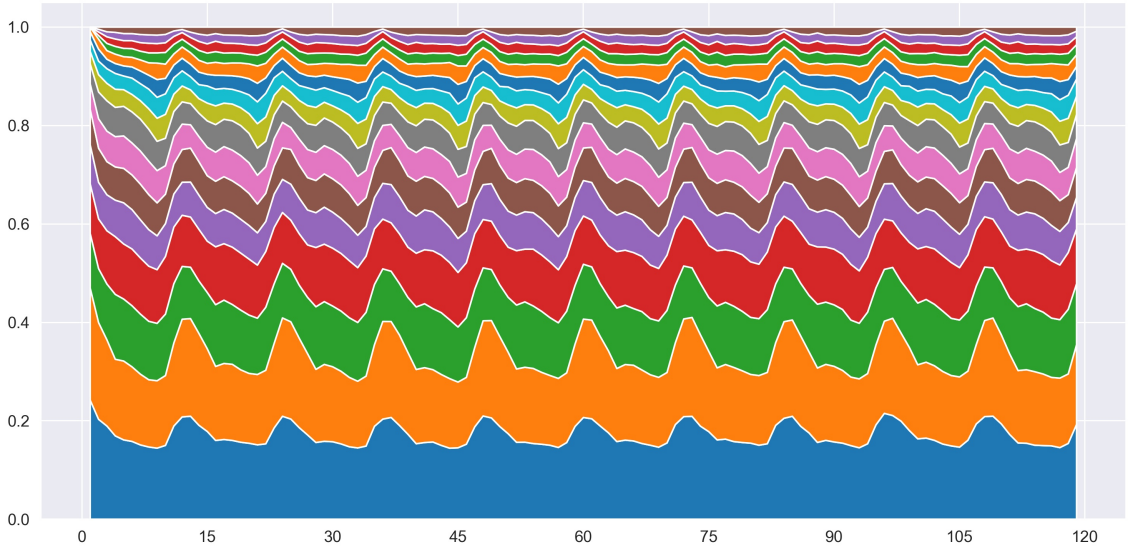
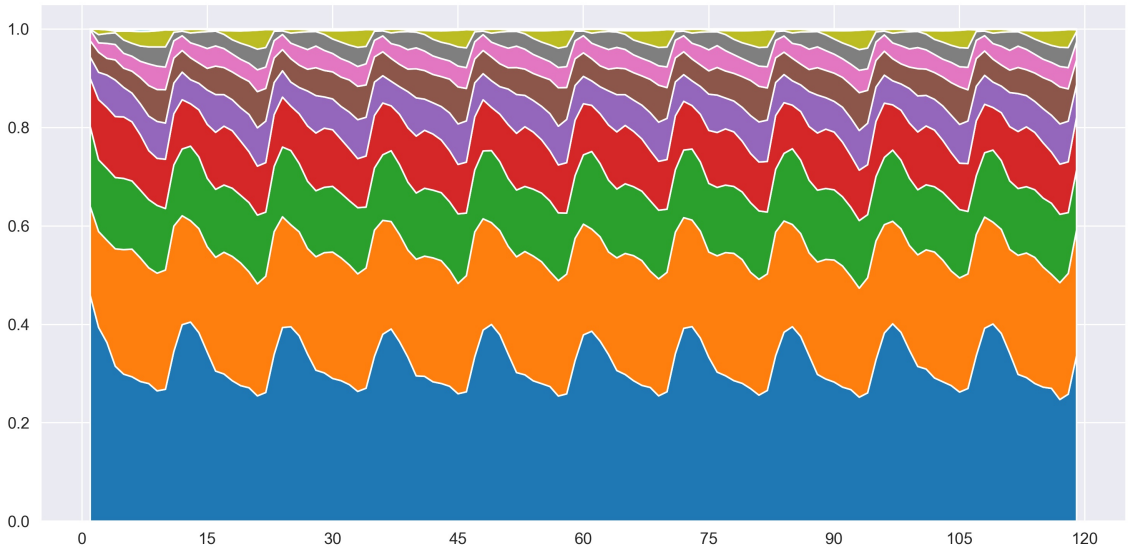


Figure 5.21: Dimension of the obtained policy when $\rho = 1$



We can also make the two systems dependent on each other. Recall that we previously make a new system by copying the original system so that each system has its own transshipment station. Now, we let the two systems share one transshipment station. In this way,

we make water exchange possible between the two systems. It turns out that it does not have much impact from an algorithmic perspective. The optimality gap after 300 iterations (again, effectively 6000 iterations) for the discretized problem is 8.11%, 16.48%, 22.94%, 28.49%, 28.85% for $\rho = 1, 0.75, 0.5, 0.25, 0$ respectively.

In the end, we investigate the effect of regularization on the SDDP algorithm for this high-dimensional problem. We consider the case that $\rho = 0$ and the two systems have their own transshipment station. We use 10 processes for parallelization. Similarly as we did in section 5.1.8, we rescale the problem to avoid numerical difficulties in solving quadratic programs. Figure 5.22 shows the evolution of lower bounds for the SDDP and the regularized SDDP with various settings of decay and weight factors. As shown, regularization seems to accelerate the SDDP algorithm a bit in all three settings for the first 100 iterations. We pick the decay and weight factors to be 0.99 and 10 respectively and continue to run the SDDP algorithm till 300 iterations. As shown in figure 5.23, the effect of regularization seems to shrink over time. This pattern is also found in section 5.1.8.

Figure 5.22: Evolution of lower bounds for 100 iterations by SDDP and regularized SDDP

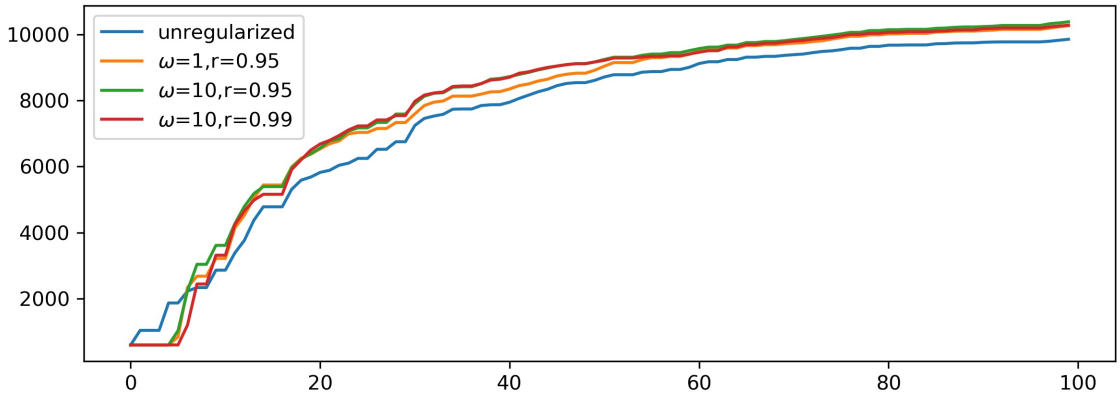
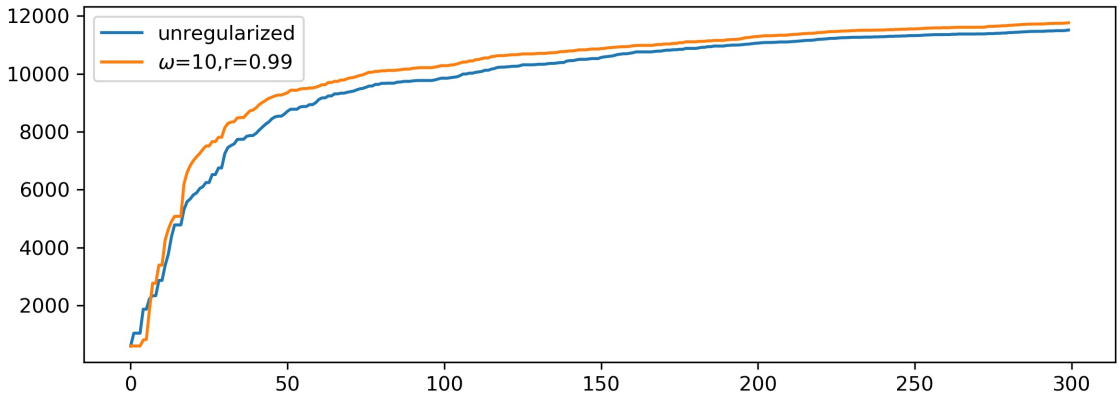


Figure 5.23: Evolution of lower bounds for 300 iterations by SDDP and regularized SDDP



5.2 Airline revenue management

5.2.1 Introduction

An airline company manages flights between three cities A, B, C. In the middle of the three cities, there is a hub H serves as a transition point. The company wants to determine the seat protection level for all itineraries, fare classes to maximize the revenue over a horizon of T stages. Every stage corresponds to a departure date. Cancellation rates Γ are deterministic. Demands are random.

$$\max \sum_{t=1}^{T} \sum_{i} \sum_{j} [f_{i,j}^{\top} b_{i,j,t} - f_{i,j}^{\top} c_{i,j,t}]$$

$$\text{s.t. } \forall t = 1, \ldots, T$$

$$B_{i,j,t} = B_{i,j,t-1} + b_{i,j,t}, C_{i,j,t} = C_{i,j,t-1} + c_{i,j,t}, \forall i, \forall j,$$

$$C_{i,j,t} = \lfloor \Gamma_{i,j} B_{i,j,t} + 0.5 \rfloor, \forall i, \forall j,$$

$$\sum_{i,j} A_{k,l}^{i,j} (B_{i,j,t} - C_{i,j,t}) \leq R_{k,l}, \forall k, \forall l,$$

$$b_{i,j,t} \leq d_{i,j,t}, \forall i, \forall j,$$

$$B_{i,j,0} = 0, C_{i,j,0} = 0, \forall i, \forall j,$$

$$b_{i,j,t}, c_{i,j,t}, B_{i,j,t}, C_{i,j,t} \in \mathbb{Z}^{+}, \forall i, \forall j.$$

The index i represents an itinerary from AH, HA, BH, HB, CH, HC, AHB, BHA, AHC, CHA, BHC, CHB. The index j represents a fare class from E1, E2, E3, E4, B1, B2 (four economy classes, two business classes). The index l represents a flight leg from AH, HA, BH, HB, CH, HC. The index k represents either a economy cabin or a business cabin. In each stage t, the current number of cumulative fulfilled booking (cancellation) requests $B_{i,j,t}(C_{i,j,t})$ is equal to its value in the previous stage $B_{i,j,t-1}(C_{i,j,t-1})$ plus the number of new fulfilled booking(cancellation) requests $b_{i,j,t}(c_{i,j,t})$. $\Gamma_{i,j}$ represents the cancellation rate. The number of new fulfilled booking requests $b_{i,j,t}$ can not be larger than the demand $d_{i,j,t}$. The capacity for each cabin and each flight leg is given by $R_{k,l}$. $A_{k,l}^{i,j}$ is one if and only if itinerary i includes flight leg l and fare class j is in cabin k; otherwise it is zero. The revenue of a fulfilled booking request is $f_{i,j}$. We assume full refund for cancellations, thus the cost of a cancellation is also $f_{i,j}$. We consider $T = 14$.

5.2.2 Modeling the demand

To model the demand process, we need to take into account two things [47], [48]. One is the uncertain total number of cumulative booking requests, the other is the uncertain arrival pattern. The non-homogenous Poisson process is a perfect model to include both of them. The expected total number of cumulative booking requests over the booking horizon $G_{i,j}$ is assumed to follow a Gamma distribution. The Gamma distribution is chosen due to empirical evidence and analytical convenience that it is the conjugate prior of the Poisson distribution. The arrival pattern of the booking requests $\beta_{i,j}(t)$ is assumed to follow a Beta distribution to allow flexible shape. The arrival intensity function of booking requests is then given by $\lambda_{i,j}(t) = \beta_{i,j}(t)G_{i,j}$. It follows that the arrival intensity of booking request $d_{i,j}(t)$ over a booking interval $[t_n, t_m]$ is $G_{i,j}[F_{\beta_{i,j}}(t_m) - F_{\beta_{i,j}}(t_n)]$. In the end, the demand process is modeled by the number of booking requests subtracting cancellation. This is achieved by scaling $G_{i,j}$ with a canceling rate. The demand process $d_{i,j}(t)$ is thus 72-dimensional and is Markovian if augmented with $G_{i,j}$. To get an idea of what the true

process looks like, we show fan plots of 1000 sample paths of simulated demands from the true process for itinerary AH and all fare classes (E1,E2,E3,E4,B1,B2) in the left column of Figure 5.24. The bell pattern as shown in the plots suggests different granularity of Markov chain discretization for different stages. We construct three discretized problems by MCA through the SA approach. The parameters of the Markov chains are rounded to the closet integers after training (since demands are integers). In Table 5.10, we provide summary of the three discretized problems. Columns 2 and 3 show the number of sample paths and time cost to train the Markov chain; the last few columns give the granularity of the Markov chain for each stage.

Table 5.10: Three airline revenue management discretized problems

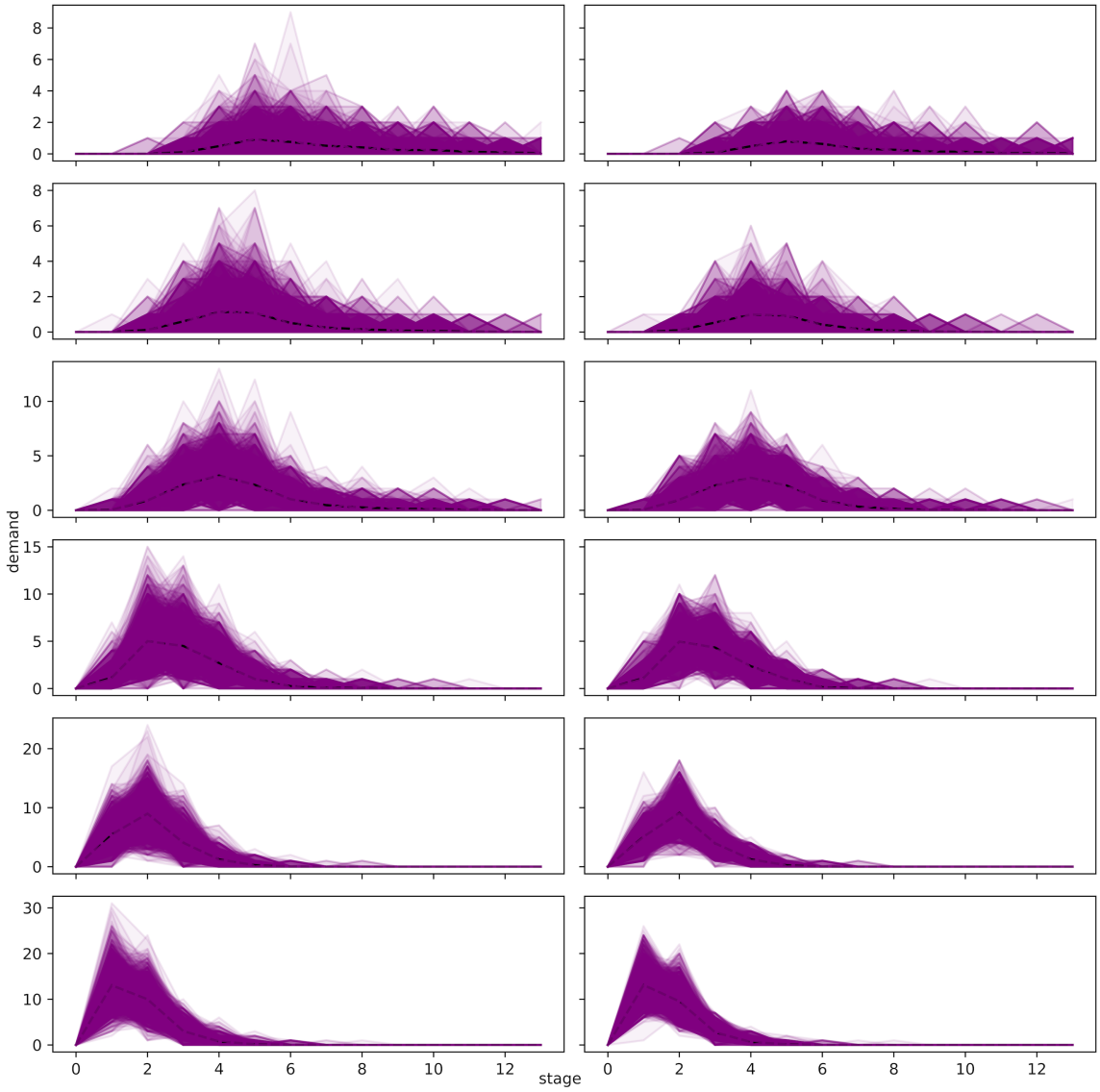
model	#iter. MCA	time	granularity of discretization for each stage						
			1	2...9	10	11	12	13	14
1	150,000	190	1	100...100	100	100	79	59	40
2	300,000	766	1	200...200	199	197	122	95	50
3	600,000	3,247	1	400...400	396	387	237	168	79

In the right column of Figure 5.24, we show fan plots of 1000 sample paths of demands simulated from discretized problem 1. The true processes and the corresponding Markov chain approximation look similar.

5.2.3 Markov chain approximation

We solve the three discretized problems using the SDDiP solver. We implement both Benders' cut and Strengthened Benders' cut for 100 iterations with a single step for each iteration. The MIP tolerance is the default value of 10^{-4}. Information about the construction and evaluation of the six policies (with 3000 Monte Carlo simulations) are summarized in table 5.11. Compared to the Benders' cut, the Strengthened Benders' cut reduces the optimality gap (column 4) for discretized problems by around 1% and it increases the expected revenue marginally (the last column). In addition, given the same type of cut, all three discretized problems provide similar expected revenues; hence, discretized problem 3 seems

Figure 5.24: Fan plots of the true demand process (on the left) and its Markov chain approximations (on the right)



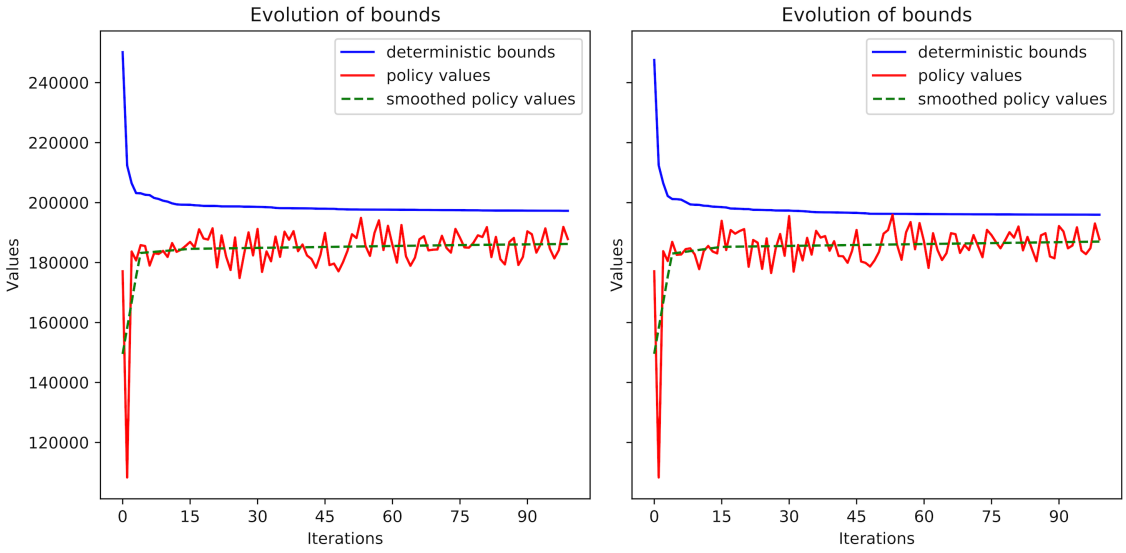
to already provide us with a reasonable good policy. Figure 5.25 visualizes the evolution of the bounds for discretized problem 3.

This experiment was done using MSPPy [1], Gurobi 8.1.0, Python 3.7.1 in macOS.

Table 5.11: Solving the three airline revenue management discretized problems and evaluation results

model	cut	time (sec.)	gap	bound ($K)	95% CI approx. ($K)		95% CI true ($K)	
1	B	807	5.74%	200.37	188.86	189.16	189.55	189.94
1	SB	5,086	4.79%	199.19	189.64	189.94	190.24	190.63
2	B	1,573	5.90%	196.48	184.89	185.23	189.50	189.88
2	SB	9,913	4.95%	195.26	185.60	185.95	190.23	190.62
3	B	3,181	5.83%	197.24	185.74	186.08	189.56	189.95
3	SB	18,232	4.80%	195.94	186.53	186.87	190.44	190.83

Figure 5.25: Evolution of bounds for the Benders' cut (on the left) and the Strengthened Benders' cut (on the right)



5.3 Multi-period portfolio optimization

5.3.1 Introduction

A portfolio manager oversees multiple assets ($i = 1, \ldots, N$) and a bank account ($i = N + 1$). For a specified number T of stages, the manager wants to maximize his utility by dynamically rebalancing the portfolio. Let $\{r_{it}\}$ be the return process of asset i. At the end of each period, the position of i^{th} asset x_{it} equals the start position $x_{i,t-1}$, plus the realized return $r_{it}x_{i,t-1}$ during the period, plus the newly long positions b_{it}, minus the newly short positions s_{it}. Transaction costs are f_b, f_s for buying and selling respectively. The capital in

the bank account will be adjusted accordingly.

$$
\begin{aligned}
&\max\ U(r_T^\top x_T)\\
&\quad \text{s.t. } \forall t = 1,\dots,T, \forall i = 1,\dots,N,\\
&\qquad x_{it} = (1+r_{it})x_{i,t-1} + x_{i,t-1} + b_{it} - s_{it},\\
&\qquad x_{N+1,t} = (1+r_{ft})x_{N+1,t-1} - (1+f_b)\sum_{i=1}^{N} b_{it} + (1-f_s)\sum_{i=1}^{N} s_{it},\\
&\qquad x_{i0} = 0, x_{N+1,0} = \$100\\
&\qquad b_{it}, s_{it}, x_{it}, x_{N+1,t} \geq 0.
\end{aligned}
$$

5.3.2 Modeling the return process

We consider a simple asset pricing model that decomposes the excess return as the return explained by Capital Asset Pricing Model (CAPM), alpha and idiosyncratic risk,

$$r_{it} = r_{ft} + \beta_i(r_{Mt} - r_{ft}) + \epsilon_{it}, \text{where } \epsilon_{it} \overset{i.i.d}{\sim} N(\alpha_i, \sigma_i),$$

where $\alpha_i, \beta_i, \sigma_i$ are assumed to be constant. We refer to $\{r_{ft} + \beta_i(r_{Mt} - r_{ft})\}$ as the market-exposure return process and $\{\epsilon_{it}\}$ as the idiosyncratic return process. The market return process $\{r_{Mt}\}$ is modeled as a first-order autoregressive process (AR) with a normal generalized autoregressive conditional heteroscedastic GARCH(1,1) innovation due to [49],

$$
\begin{aligned}
&r_{Mt} = \mu + \phi r_{M,t-1} + \epsilon_{Mt},\\
&\epsilon_{Mt} = \sigma_{Mt} e_{Mt},\\
&\sigma_{Mt}^2 = \omega + \alpha_1 \epsilon_{M,t-1}^2 + \alpha_2 \sigma_{M,t-1}^2,\\
&e_{Mt} \overset{i.i.d}{\sim} \mathcal{N}(0,1).
\end{aligned}
$$

Note that the above model can be further improved. For example, the considered asset pricing model fails to incorporate important factors such as SMB (small market capitalization minus big) and HML (high book-to-market ratio minus low). A more realistic asset pricing model such as [50] can be considered. Furthermore, empirical evidence such as [51] have found that the normal GARCH innovation fails to capture the leptokurtosis and heavy tails in financial time series. Nevertheless, from now on, we will assume that the AR(1)-GARCH(1,1) is our "true process". We consider $K = 100$ and $T = 12$.

5.3.3 Markov chain approximation

The resulting true problem has mixed types of data processes and should be discretized accordingly. The stage-wise independent idiosyncratic return processes $\{\epsilon_{it}\}$ are discretized by SAA with 100 i.i.d. samplings. The Markovian market-exposure return processes $\{r_{ft} + \beta_i(r_{Mt} - r_{ft})\}$ are non-linear and thus should be discretized by MCA rather than the TS approach. Given the fact that the market-exposure return processes are virtually determined by the market return process, we can first discretize the market return process and then arithmetically compute the discretization of the market-exposure return process.

We set the granularity of Markov chain discretization to be 100 for all stages except the first stage. As shown in Figure 5.26 and Table 5.12, the Markov chain process gives fairly good approximation of the true market return process. Most of the statistics, including expected return (sample mean), risk (sample standard deviation), 5% value at risk, skewness are similar. But it fails to fully resemble the kurtosis and tails. This behavior is expected since the Markov states are the centers of partitions of the sample space and are not likely to cover the corner of the space.

We obtain five policies by solving the constructed discretized problem for 50 iterations and 3 steps per each iteration under risk measures of different combinations of expectation and AVaR. In particular, the risk neutral discretized problem is solved with a 2.19% optimality gap. Table 5.13 and Table 5.14 gives the evaluation results for these policies with

Figure 5.26: Fan plots of the true return process and its Markov chain approximation through SA

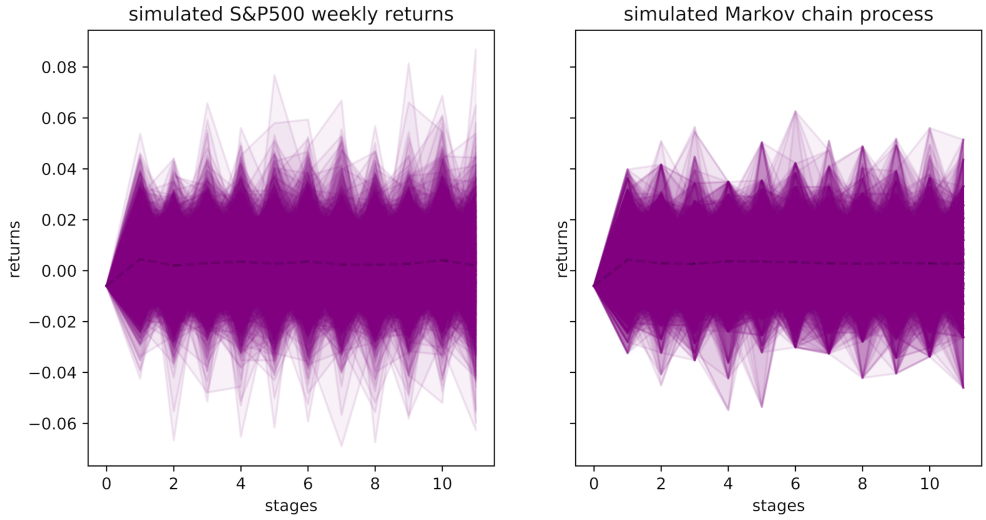


Table 5.12: Statistics of the cumulative return of the true market return process versus that of its approximating Markov chain process over the whole period (kurtosis is calculated as the fourth standardized moment. For example, kurtosis of the normal distribution is 3).

	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
True process	2.813	4.791	4.671	0.248	3.707	0.47
Markov chain	2.993	4.916	4.926	0.210	3.311	0.50

1000 Monte Carlo simulations, in which λ represents the weight of AVaR and α represents the level of VaR. The introduction of risk measures changes the distribution of the final wealth dramatically. Policy $\lambda(0.75) - \alpha(0.25)$ almost gives a riskless allocation scheme that keeps allocating all the money in the bank account (the final return of a riskless strategy is $(1+0.05\%)^{11}-1 = 0.551\%$). If we use the return-volatility ratio (the last column) as the performance evaluation metrics, policy $\lambda(0.25)-\alpha(0.25)$ seems to be a promising strategy that beats a passive strategy holding the (simulated) market index. If we also take value-at-risk (column 4) into consideration, policy $\lambda(0.50) - \alpha(0.25)$ is most desirable since it offers both a slightly bigger Sharpe Ratio and a lower value-at-risk than the market index.

This experiment was done using MSPPy [1], Gurobi 8.1.0, Python 3.7.1 in macOS.

Table 5.13: Policy evaluation for the portfolio optimization discretized problem

policy	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
Risk neutral	16.538	22.071	17.410	0.538	3.664	0.75
$\lambda(0.25) - \alpha(0.25)$	10.285	6.960	1.068	0.043	2.893	1.48
$\lambda(0.50) - \alpha(0.25)$	4.298	3.392	1.570	-0.205	3.558	1.27
$\lambda(0.75) - \alpha(0.25)$	0.551	<0.001	-0.551	0.101	3.691	/

Table 5.14: Policy evaluation for the portfolio optimization true problem

policy	expected return (%)	risk (std)	5%VaR ($)	skewness	kurtosis	Sharpe Ratio
Risk neutral	14.425	22.045	19.327	0.407	3.180	0.63
$\lambda(0.25) - \alpha(0.25)$	6.137	7.259	5.756	-0.166	3.595	0.77
$\lambda(0.50) - \alpha(0.25)$	2.640	3.680	3.980	-0.370	3.441	0.57
$\lambda(0.75) - \alpha(0.25)$	0.551	<0.001	-0.551	-0.029	3.816	/

5.4 Hedging

5.4.1 Introduction

Suppose the price process of the hedged instrument is F_t and the time to maturity is T. The hedging instruments are assumed to be cash (C) and stock (S). The goal is to construct a tracking portfolio (V) that minimizes the hedging error,

$$
\begin{aligned}
\min\ & \mathbb{E}\big[\sum_{t=1}^{T} |V_t - F_t|\big] \\
\text{s.t. } & V_t = S_t + C_t \\
& S_t = S_{t-1} + b_t - s_t \\
& C_t = (1 + r_t)C_{t-1} + (1 - h_t)s_t - (1 + h_t)b_t \\
& b_t \geq 0, s_t \geq 0, S_0 = 0, C_0 \text{ is given},
\end{aligned}
$$

where $b_t(s_t)$ represents dollar amount of stock bought(sold) at time t, S_t and C_t denotes the position of stock and cash, and h_t is the proportional transaction fee. We consider hedging an at-the-money European call option for 7 days or 7 years. Portfolio is allowed to be

rebalanced once a day or once a year. The proportional transaction fee is assumed to be 0%, 0.1% or 1%. We assume the stock price X_t follows a geometric Brownian motion.

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

The stock price process therefore has the analytic solution $S_t = S_0 \exp((\mu - \frac{\sigma^2}{2})t + \sigma W_t)$. The price process of the at-the-money European option is then given by Black Scholes formula,

$$\begin{aligned} F_t &= S_0 N(d_1) - K e^{-r\tau} N(d_2) \\ d_{1,2} &= \frac{(r \pm \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}} \\ \tau &= T - t, \end{aligned}$$

where $N(\cdot)$ is the cumulative distribution function of the standard normal.

Figure 5.27 shows the trajectory of the simulated stock prices for 7 days (on the left) and 7 years (on the right).

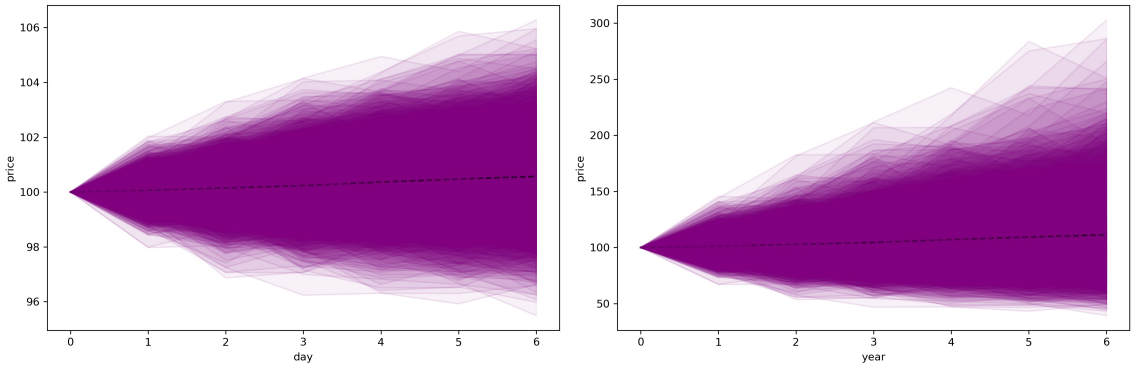


Figure 5.27: Trajectory of simulated stock prices

5.4.2 Rolling horizon algorithm v.s. SDDP algorithm

We use Markov chain approximation to discretize the stock price process and then employ the Markov chain SDDP to solve the discretized problem. Alternatively, we can simply use the rolling horizon algorithm to solve this problem. We will refer to these two approaches as the mc hedging strategy and the rolling hedging strategy respectively. The traditional delta hedging strategy is used as the benchmark of this analysis.

Table 5.15: Comparison of delta hedging, mc-hedging and rolling hedging

		0				0.1%				1%			
		min	max	left	right	min	max	left	right	min	max	left	right
7-day	delta	0.06	4.15	0.64	0.67	0.06	4.84	0.72	0.76	2.16	11.79	4.96	5.05
	rolling	0.04	8.59	0.62	0.67	0.02	8.29	0.66	0.73	0.08	20.52	2.35	2.56
	mc-51	0.04	38.14	0.63	0.71	0.02	8.68	0.61	0.67	0.05	18.93	2.31	2.50
	mc-101	0.04	10.37	0.60	0.64	0.01	22.35	0.60	0.67	0.03	18.99	2.31	2.50
7-year	delta	0.48	63.31	9.14	9.59	0.61	64.25	8.96	9.42	0.72	72.64	9.57	10.18
	rolling	0.53	167.75	8.58	9.16	0.20	181.48	6.60	7.28	0.39	173.64	7.90	8.69
	mc-51	0.76	176.77	8.45	8.99	0.27	189.44	6.24	6.86	0.27	315.77	7.67	8.53
	mc-101	0.76	184.69	8.40	8.94	0.30	183.30	6.13	6.75	0.30	284.57	7.53	8.35

Table 5.15 summarizes the evaluation result on the true problem for these two approaches. In each panel, the four columns represent the minimum value, the maximum value, and left and right ends of the confidence interval of the simulated hedging error respectively (obtained by 3000 Monte carlo simulations). We vary the number of Markov states by 51 and 101, corresponding to mc-51 hedging strategy and mc-101 hedging strategy respectively. From the table, we can make the following observations. First, as the proportional transaction costs increase, the traditional delta hedging strategy performs worse as opposed to the other three strategies. This is expected because the traditional delta hedging strategy does not take into account the transaction costs. Second, the mc hedging in general outperforms the rolling hedging. The only exception occurs in the 7-day hedging problem, where the mc-51 hedging underperforms the rolling hedging, suggesting that the corresponding Markov chain approximation is not granular enough. Indeed, the more granular strategy, the mc-101 hedging, seems to perform better than the rolling hedging. This kind of behaviour is also seen in section 5.1.9.

This experiment was done using MSPPy [1], Gurobi 8.1.0, Python 3.7.1 in Red Hat Linux.

5.5 Comparing MSPPy with other software packages

Throughout previous sections, we have seen that a crucial step is to evaluate the policy of the true problem. As far as we know, none of the existing software packages provides good answer to that. All of them starts with formulation of a finite discrete problem as (2.9) or (2.10). In other words, they make the assumption that the number of scenarios is finite. In real world applications, finiteness barely holds. In other cases where the number of scenarios is finite but huge, true problem is intractable and also needed to be discretized. Consequently, the existing software packages will only be able to solve discretized problems. However, solving the discretized problem, even to optimality, guarantees nothing about its performance for the true problem. It also raises another important question and makes it more problematic. What is a proper way of discretization to get a reasonable discretized problem in the first place? A proper way may include guidance on how granular should the discretization be and how to chose among different discretization techniques. Unfortunately, the existing software packages will not help you with that. Therefore, to understand the whole picture of a multistage stochastic program, users of existing software packages should create their own interface to make discretization and to evaluate policy on their true problem. As a result, they have to switch interfaces back and forth, which is highly inconvenient.

The main innovation of the MSPPy package is that it offers the all-in-one functionality to build and discretize the true problem, solve the constructed approximation problem and evaluate the computed policy on the true problem in the same interface. Users are able to clearly understand how policies perform on the true problem and choose the best among them through statistical analysis as we did in the previous sections. The MSPPy package is also the first one to implement a rolling solver (based on rolling horizon algorithm) and

a PSDDP/PSDDiP solver (based on periodical SDDP/SDDiP).

The MSPPy package chooses Python as its main interface since Python is one of the sky-rocketing programming languages. The abundant analytic tools and libraries make optimization, statistical analysis, and visualization very easy. The MSPPy package leverages the NumPy package to make fast numerical computation and the Pandas package to make visualization and output readable result. Gurobi [26] is chosen as the external solver since some empirical evidence have shown, Gurobi solver performs better than other solvers in various cases, e.g., [52]. While the binding with Gurobi also brings in certain limitations on the types of subproblems the MSPPy package can solve. Given the fact that up to now Gurobi solver is not able to solve generic convex programs, the MSPPy package only supports LP, MIP, QP subproblems. On the other hand, SDDP.jl leverages the JuMP package [53], which provides a generic interface of various solvers. As a result, SDDP.jl and its extension SDDiP.jl can also solve generic convex subproblems. Nevertheless, it is very rare to have a non-quadratic convex subproblems.

The MSPPy package largely inherits the syntax in the Gurobi python interface. Hence the high-level modeling capability of Python is retained. SDDP.jl, on the other hand, leverages the macro-programming provided by JuMP, has a different modeling benefit. We compare the performance of SDDP.jl and MSPPy by implementing the simplified version of the Brazilian hydro-thermal power system example introduced in section 5.1.

For comparison convenience, here we simply model energy inflows as stage-wise independent discrete stochastic processes. The sample space is composed of historical monthly inflows (around 100). For simplicity, The tiny costs for water spillage and water exchanges are ignored. The discount factor is set to be 1.

Table 5.16 gives implementation result of the problem after running both the packages for 200 seconds for repetitively ten times. In column 2, we report the number of processes the solver use; column 3 shows the average lower bounds. As shown, the MSPPy package is more efficient than SDDP.jl in this case. Part of the reasons is because the cost of additional

Table 5.16: Comparison of MSPPy and SDDP.jl

solver	# processes	lower bound ($M)
SDDP.jl	1	291.8
SDDP.jl	3	292.6
MSPPy	1	293.1
MSPPy	3	294.4

abstraction layer JuMP includes to adapt to various solvers. This implementation was done using Python 3.5.0, Julia 0.6 and Gurobi 7.0.2 on a Mac Pro (in macOS Sierra 10.12.6) with a 2.7 GHz Intel Core i5-5257U processor.

REFERENCES

[1] L. Ding, S. Ahmed, and A. Shapiro, "A python package for multi-stage stochastic programming," *Optimization online*, 2019.

[2] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[3] V. Guigues, A. Shapiro, and Y. Cheng, "Duality and sensitivity analysis of multistage linear stochastic programs," *arXiv preprint arXiv:1911.07080*, 2019.

[4] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[5] V. Bally and G. Pages, "A quantization algorithm for solving multidimensional discrete-time optimal stopping problems," *Bernoulli*, vol. 9, no. 6, pp. 1003–1049, 2003.

[6] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[7] M. V. Pereira and L. M. Pinto, "Multi-stage stochastic optimization applied to energy planning," *Mathematical programming*, vol. 52, no. 1-3, pp. 359–375, 1991.

[8] G. Lan, "Complexity of stochastic dual dynamic programming," *arXiv preprint arXiv:1912.07702*, 2019.

[9] S. Zhang and X. A. Sun, "Stochastic dual dynamic programming for multistage stochastic mixed-integer nonlinear optimization," *arXiv preprint arXiv:1912.13278*, 2019.

[10] A. Shapiro, "Analysis of stochastic dual dynamic programming method," *European Journal of Operational Research*, vol. 209, no. 1, pp. 63–72, 2011.

[11] A. Ruszczyński and A. Świętanowski, "Accelerating the regularized decomposition method for two stage stochastic linear problems," *European Journal of Operational Research*, vol. 101, no. 2, pp. 328–342, 1997.

[12] T. Asamov and W. B. Powell, "Regularized decomposition of high-dimensional multistage stochastic programs with markov uncertainty," *SIAM Journal on Optimization*, vol. 28, no. 1, pp. 575–595, 2018.

[13] A. I. Mahmutoğulları, S. Ahmed, Ö. Çavuş, and M. S. Aktürk, "The value of multistage stochastic programming in risk-averse unit commitment under uncertainty," *IEEE Transactions on Power Systems*, vol. 34, no. 5, pp. 3667–3676, 2019.

[14] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, "Coherent measures of risk," *Mathematical Finance*, vol. 9, pp. 203–228, 1999.

[15] H. Föllmer and A. Schied, *Stochastic Finance: An Introduction in Discrete Time*, 2nd. Walter de Gruyter, Berlin, 2004.

[16] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, 2nd. Philadelphia: SIAM, 2014.

[17] L. Ding and A. Shapiro, "Upper bound for optimal value of risk averse multistage problems," Technical Report, Tech. Rep., 2016.

[18] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, "New variants of bundle methods," *Mathematical programming*, vol. 69, no. 1-3, pp. 111–147, 1995.

[19] J. Zou, S. Ahmed, and X. A. Sun, "Stochastic dual dynamic integer programming," *Mathematical Programming*, pp. 1–42, 2018.

[20] O. Dowson and L. Kapelevich, "Sddp. jl: A julia package for stochastic dual dynamic programming," *Optimization Online*, 2017.

[21] B Legat, "Structdualdynprog. jl, 2018,"
url: https://github. com/blegat/StructDualDynProg. jl, 2018.

[22] V. Leclère, H. Gérard, F. Pacaud, and T. Rigaut, "Stochdynamicprogramming. jl a julia library for multistage stochastic optimization," 2018.

[23] L. Cambier, "Fast," *URL https://github.com/leopoldcambier/FAST*, 2018.

[24] H. Gevret, N. Langrené, J. Lelong, X. Warin, and A. Maheshwari, "Stochastic optimization library in c++," PhD thesis, EDF Lab, 2018.

[25] L. Kapelevich, "Sddip," *URL https://github.com/lkapelevich-/SDDiP.jl*, 2018.

[26] L. Gurobi Optimization, "Gurobi optimizer reference manual,
url: Http://www.gurobi.com," 2018.

[27] J. E. Kelley Jr, "The cutting-plane method for solving convex programs," *Journal of the society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[28] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[29] A. Shapiro and L. Ding, "Stationary multistage programs," *Optimization Online*, 2019.

[30] A. Shapiro, W. Tekaya, J. da Costa, and M. P. Soares, "Risk neutral and risk averse stochastic dual dynamic programming method," *European Journal of Operational Research*, vol. 224, pp. 375–391, 2013.

[31] D. Bertsekas and S. Shreve, *Stochastic Optimal Control, The Discrete Time Case*. Academic Press, New York, 1978.

[32] R. T. Rockafellar and R. J.-B. Wets, *Variational Analysis*. New York: Springer, 1998.

[33] A. Shapiro, "Consistency of sample estimates of risk averse stochastic programs," *Journal of Applied Probability*, vol. 50, pp. 533–541, 2013.

[34] J. Warrington, N. Beuchat, and J. Lygeros, "Generalized dual dynamic programming for infinite horizon problems in continuous state and action spaces," *IEEE Transactions on Automatic Control*, 2019.

[35] M. Pereira and L. Pinto, "Multi-stage stochastic optimization applied to energy planning," *Mathematical programming*, vol. 52, no. 1-3, pp. 359–375, 1991.

[36] R. Baucke, "An algorithm for solving infinite horizon Markov dynamic programmes," *Optimization online*, 2018.

[37] A. Philpott, V. de Matos, and E. Finardi, "On solving multistage stochastic programs with coherent risk measures," *Operations Research*, vol. 61, no. 4, pp. 957–970, 2013.

[38] P. Girardeau, V. Leclere, and A. B. Philpott, "On the convergence of decomposition methods for multistage stochastic convex programs," *Mathematics of Operations Research*, vol. 40, pp. 130–145, 2016.

[39] V. Guigues, "Dual dynamic programing with cut selection: Convergence proof and numerical experiments," *European Journal of Operational Research*, vol. 258, 2017.

[40] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares, "Risk neutral and risk averse stochastic dual dynamic programming method," *European journal of operational research*, vol. 224, no. 2, pp. 375–391, 2013.

[41] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares, "Report for technical cooperation between georgia institute of technology and ons-operador nacional do sistema elétrico," 2011.

[42] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares, "Final report for technical cooperation between georgia institute of technology and ons-operador nacional do sistema eletrico," *Georgia Tech ISyE Report*, 2012.

[43] N. Löhndorf and A. Shapiro, "Modeling time-dependent randomness in stochastic dual dynamic programming," *European Journal of Operational Research*, vol. 273, no. 2, pp. 650–661, 2019.

[44] A. Möller, W. Römisch, and K. Weber, "Airline network revenue management by multistage stochastic programming," *Computational Management Science*, vol. 5, no. 4, pp. 355–377, 2008.

[45] G. B. Dantzig and G. Infanger, "Multi-stage stochastic linear programs for portfolio optimization," *Annals of Operations Research*, vol. 45, no. 1, pp. 59–76, 1993.

[46] M. Villaverde, "Hedging european and barrier options using stochastic optimization," *Quantitative Finance*, vol. 4, no. 5, pp. 549–557, 2004.

[47] L. Chen and T. Homem-de Mello, "Re-solving stochastic programming models for airline revenue management," *Annals of Operations Research*, vol. 177, no. 1, 2010.

[48] S. V. de Boer, R. Freling, and N. Piersma, "Mathematical programming for network revenue management revisited," *European Journal of Operational Research*, vol. 137, no. 1, pp. 72–92, 2002.

[49] V. Akgiray, "Conditional heteroscedasticity in time series of stock returns: Evidence and forecasts," *Journal of business*, pp. 55–80, 1989.

[50] E. F. Fama and K. R. French, "A five-factor asset pricing model," *Journal of financial economics*, vol. 116, no. 1, pp. 1–22, 2015.

[51] T. Bollerslev *et al.*, "A conditionally heteroskedastic time series model for speculative prices and rates of return," *Review of economics and statistics*, vol. 69, no. 3, pp. 542–547, 1987.

[52] I Gurobi Optimization, "Gurobi 7.5 performance benchmarks," 2018.

[53] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.